NASA-CR-193048

NAS2-13296

Intelligent Systems Engineering

IN-12-CR
163068

Final Report

Knowledge-Based Decision Support for Space Station Assembly Sequence Planning

ISX Corporation
4353 Park Terrace Drive
Westlake Village, CA 91361

(NASA-CR-193048) KNOWLEDGE-BASED DECISION SUPPORT FOR SPACE STATION ASSEMBLY SEQUENCE PLANNING Final Report (ISX Corp.) 142 p

N93-26095

Unclas

G3/12 0163068

Final Report

Knowledge-Based Decision Support for Space Station Assembly Sequence Planning

Submitted By

ISX Corporation
4353 Park Terrace Drive
Westlake Village, CA 91361

April 12, 1991

Funded By

NASA Ames Research Center
NASA Research Announcement Contract NAS2-13296

Table of Contents

1.0 Introduction

1.1 Purpose and Scope

This is the final report on work performed under NASA Ames Research Center (ARC) Research Announcement Contract NAS2-13296.

1.2 Identification

This is the first released version of the final report on work performed under NASA ARC Research Announcement Contract NAS2-13296. It is designated version 1.0 and is dated April 12, 1991.

1.3 Contents

Section 2 lists references for products produced in this project. Section 3 presents a project summary. An overview of plans to continue development of an assembly sequence planning system beyond the contracted work is presented in Section 4. Detailed information of project objectives, work carried out, and results are presented in the documents listed in Section 2.

2.0 References

Application Assessment Report: Space Station Assembly Sequence Planning. ISX Corporation. March 6, 1990.

Requirements Specification: Knowledge-Based Decision-Support System for SSF Engineering Managers. Version 1.0, ISX Corporation, May 11, 1990.

Preliminary Design Document: Knowledge-Based Decision-Support System for SSF Engineering Managers. Version 1.0, ISX Corporation, June 15, 1990.

Assembly Sequence Planning Personal Analysis Assistant: User's Reference Guide. Version 1.0, ISX Corporation, February 14, 1991.

3.0 Project Summary

This work is funded by NASA ARC Research Announcement Contract NAS2-13296. The objective is the phased development of a Personal Analysis Assistant for SSF assembly sequence planning. Expertise was provided by Space Station Freedom Program Office (SSFPO) staff and contractors.

A complete Personal Analysis Assistant (PAA) for SSF assembly sequence planning consists of three software components: the System Infrastructure, Intra-Flight Value Added, and Inter-Flight Value Added. The System Infrastructure is the substrate on which software elements providing inter-flight and intra-flight value-added functionality are built. It provides the capability for building representations of assembly sequence plans and specification of constraints and analysis options. Intra-Flight Value-Added provides functionality that will, given the manifest for each flight, define cargo elements, place them in the NSTS cargo bay, compute performance measure values, and identify violated constraints. Inter-Flight Value-Added provides functionality that will, given major

Copyright © 1991, ISX Corporation

milestone dates and capability requirements, determine the number and dates of required flights and develop a manifest for each flight. The current project is Phase 1 of a projected two phase program and delivers the System Infrastructure. Intra- and Inter-Flight Value-Added were to be developed in Phase 2, which has not been funded.

Based on experience derived from hundreds of projects conducted over the past seven years, ISX has developed an Intelligent Systems Engineering (ISE) methodology that combines the methods of systems engineering and knowledge engineering to meet the special systems development requirements posed by intelligent systems, systems that blend artificial intelligence and other advanced technologies with more conventional computing technologies. The ISE methodology defines a phased program process that begins with an application assessment designed to provide a preliminary determination of the relative technical risks and payoffs associated with a potential application, and then moves through requirements analysis, system design, and development.

The Application Assessment. The assessment indicated the value and feasibility of a "Personal Analysis Assistant" that would perform the work that is currently so tedious and time-consuming for the human assembly sequence planner. The document "Application Assessment Report: Space Station Assembly Sequence Planning," cited in Section 2, presents assessment results in detail.

The Requirements Analysis. The requirements analysis for the Personal Analysis Assistant followed the application assessment. The document "Requirements Analysis: Knowledge-Based Decision-Support System for SSF Engineering Managers," cited in Section 2, describes the results of the requirements analysis.

System Design. The system design extended and generalized the system specification begun in the requirements specification. The document "Preliminary Design Document: Knowledge-Based Decision-Support System for SSF Engineering Managers," cited in Section 2, describes the the system design.

System Development. As noted above, this project is Phase 1 of a projected two phase program. The system development deliverable of Phase 1 is the System Infrastructure component of the complete Personal Analysis Assistant. The results of Phase 1 system development are described in the document "Assembly Sequence Planning Personal Analysis Assistant: User's Reference Guide" which is cited in Section 2.

4.0 Phase 2 Plans

Phase 2 of the projected two-phase program was to deliver Intra- and Inter-Flight Value-Added functionality, a Cargo Bay Editor, and several user-interface enhancements. This phase has not been funded and at the present time there are no plans for the development of these components.

The documents "Requirements Analysis: Knowledge-Based Decision-Support System for SSF Engineering Managers" and "Preliminary Design Document: Knowledge-Based Decision-Support System for SSF Engineering Managers," both of which are cited in Section 2, present descriptions of the Intra- and Inter-Flight Value-Added functionality that would have been provided in Phase 2. Intra- and Inter-Flight Value-Added functionality is derived from an "Associate System" concept of man-machine interaction that has evolved from several years of government-sponsored research and development in a variety of application domains, including the Pilot's Associate, an advanced pilot aid

Copyright © 1991, ISX Corporation

developed for DARPA. The associate system operates like a colleague who helps the manager perform his or her work. Like a good human associate, the system monitors the status of the task in order to provide the level of support appropriate to the user's current information needs. The Phase 2 Personal Analysis Assistant would have provided support to the assembly sequence planner in determining the number and dates of required flights, developing a manifest for each flight, defining cargo elements, placing cargo elements in the NSTS cargo bay, computing performance measure values, and identifying violated constraints.

Development of Intra- and Inter-Flight Value-Added components would have been supported by MAX, the "Manager's Associate," a framework for developing associate systems developed under a NASA ARC Phase II SBIR contract.

The Cargo Bay Editor was to have been a graphic user-interface supporting placement of cargo bay elements in the NSTS cargo bay by direct manipulation of icons and immediate display of constraint values by means of Kohr's Curve and other graphic displays. Figure 1 shows a notional design for the interface of the Cargo Bay Editor.

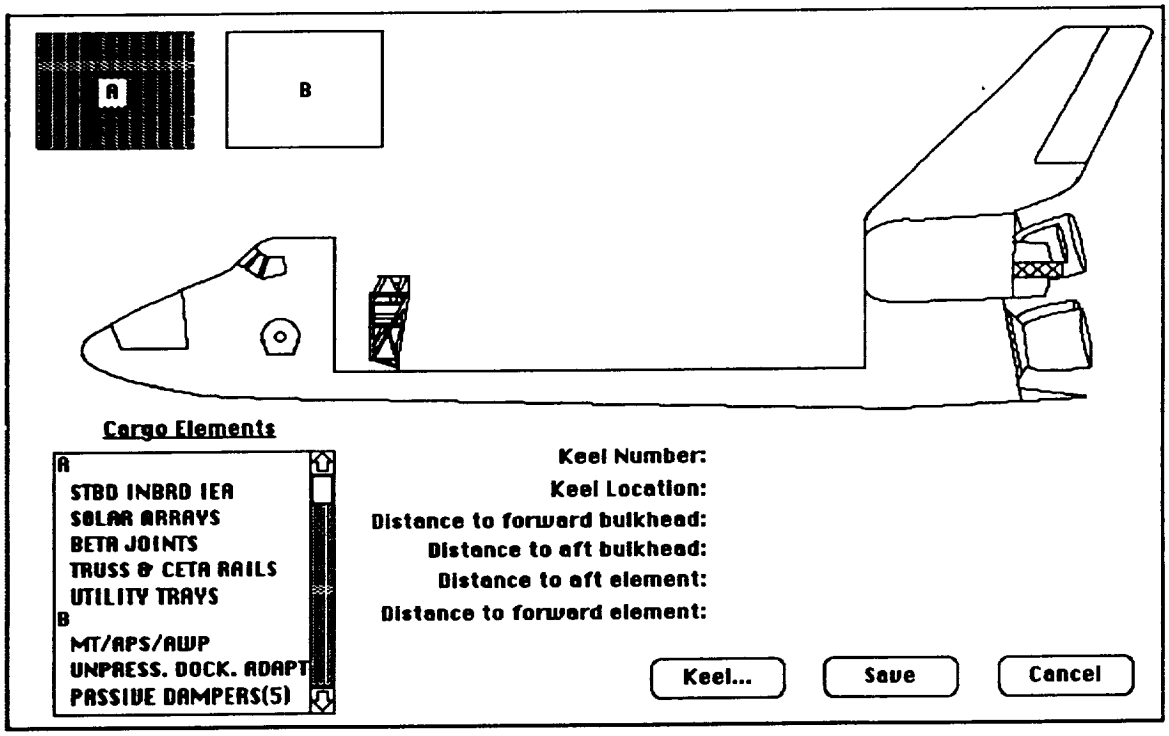


Figure 1: Notional Design for the Cargo Bay Editor User Interface

The Cargo Bay Editor would be operated as follows. The upper left corner of the editor displays the envelopes containing the mission's cargo elements. Selecting a cargo element causes a dialog to appear which invites the user to specify the element's geometric attributes, including its envelope, keel pin and trunnion point locations, cg location, and whether it is a full or half height element. After an element has been specified, it may be dragged into the cargo bay for placement. To do that, the user selects the element and, with the mouse, drags the cargo element to the cargo bay. When the element reaches the bottom of the bay, it will "click" into place at the nearest keel pin/trunnion location which matches the element's configuration. As the user slides the element along the bottom of the

Copyright © 1991, ISX Corporation

bay, the element continues to "click" into the next acceptable location. As the element approaches another element, or the forward or aft cargo bay bulkhead, an alert is displayed when the element violates the buffer distance constraint. The user then has the option of overriding the violated constraint or returning the element to the last acceptable location. The user may overlap elements to provide for the case where an element has a convex face which complements a concave face of a adjacent element. The user may also reverse the orientation of an element, such that the element's forward face becomes the aft face. Directly below the cargo bay are a series of text fields which provide dynamic information about the state of the currently selected element and the center of gravity margin as determined by the Kohr's Curve. A description of each cargo element is shown in the lower left corner of the editor while the "Save", "Cancel", and "Keel..." buttons are located in the lower right. Pressing the "Keel..." button displays a dialog which enables the user to alter the configuration of the keel pin/trunnion locations.

Copyright © 1991, ISX Corporation

Final Report

Knowledge-Based Decision-Support System for SSF Engineering Managers

Submitted By

ISX Corporation
501 Marin St., Suite 214
Thousand Oaks, CA 91360

June 15, 1990

Funded By

NASA Ames Research Center
Small Business Innovative Research Contract NAS2-13161

Table of Contents

1.0 Introduction

1.1 Purpose and Scope

This is the final report on work performed under NASA Ames Research Center (ARC) Phase I SBIR (Small Business Innovative Research) Contract NAS2-13161.

1.2 Identification

This is the first released version of the final report on work performed under NASA ARC Phase I SBIR Contract NAS2-13161. It is designated version 1.0 and is dated June 15, 1990.

1.3 Contents

Section 2 lists references for products produced in this SBIR. Section 3 presents a project summary. A general discussion of plans to continue development to the fielding of a commercial product are discussed in Section 4. Detailed information of project objectives, work carried out, and results are presented in the documents listed in Section 2.

2.0 References

Application Assessment Report: Space Station Assembly Sequence Planning. ISX Corporation. March 6, 1990.

Requirements Specification: Knowledge-Based Decision-Support System for SSF Engineering Managers. Version 1.0, ISX Corporation, May 11, 1990.

Preliminary Design Document: Knowledge-Based Decision-Support System for SSF Engineering Managers. Version 1.0, ISX Corporation, June 15, 1990.

©ISX Corporation

3.0 Project Summary

This work is funded by ARC Phase I SBIR Contract NAS2-13161. The objective is the development of requirements and a preliminary design for a knowledge-based decision-support system for Space Station Freedom (SSF) engineering managers. The initial application is SSF assembly sequence planning. Expertise was provided by Space Station Freedom Program Office (SSFPO) staff and contractors.

Based on experience derived from hundreds of projects conducted over the past seven years, ISX has developed an Intelligent Systems Engineering (ISE) methodology that combines the methods of systems engineering and knowledge engineering to meet the special systems development requirements posed by *intelligent systems*, systems that blend artificial intelligence and other advanced technologies with more conventional computing technologies. The ISE methodology defines a phased program process that begins with an application assessment designed to provide a preliminary determination of the relative technical risks and payoffs associated with a potential application, and then moves through requirements analysis, system design, and development.

The Application Assessment. The assessment indicated the value and feasibility of a "Personal Analysis Assistant" that would perform the work that is currently so tedious and time-consuming for the human assembly sequence planner. The document "Application Assessment Report: Space Station Assembly Sequence Planning," cited in Section 2, presents assessment results in detail.

The Requirements Analysis. The requirements analysis for the Personal Analysis Assistant followed the application assessment. The document "Requirements Analysis: Knowledge-Based Decision-Support System for SSF Engineering Managers," cited in Section 2, describes the results of the requirements analysis. These results can be briefly summarized as follows. The Personal Analysis Assistant (PAA) consists of three software applications: the System Infrastructure, Intra-Flight Value Added, and Inter-Flight Value Added. The *System Infrastructure* is the substrate on which software elements providing inter-flight and intra-flight value-added functionality are built. It provides the capability for building representations of assembly sequence plans, constraint networks describing bounds on measures, and user-specifications of constraints and analysis options. *Intra-Flight Value-Added* provides functionality that will, given the manifest for each flight, define cargo elements, place them in the NSTS cargo bay, compute performance measure values, and identify violated constraints. *Inter-Flight Value-Added* provides functionality that will, given major milestone dates and capability requirements, determine the number and dates of required flights and develop a manifest for each flight.

System Design. The preliminary system design extended and generalized the system specification begun in the requirements specification. The generalization enables the development of decision-support systems applicable to problems other than SSF assembly sequence planning.

This work has produced requirements and a preliminary design for a framework supporting the development of a family of "assistant" systems in several problem domains. These assistant systems help the human planner by doing the bookkeeping to maintain plan data and executing the procedures and heuristics currently used by the human planner to define, assess, diagnose, and revise plans.

©ISX Corporation

4.0 Toward a Commercial Product

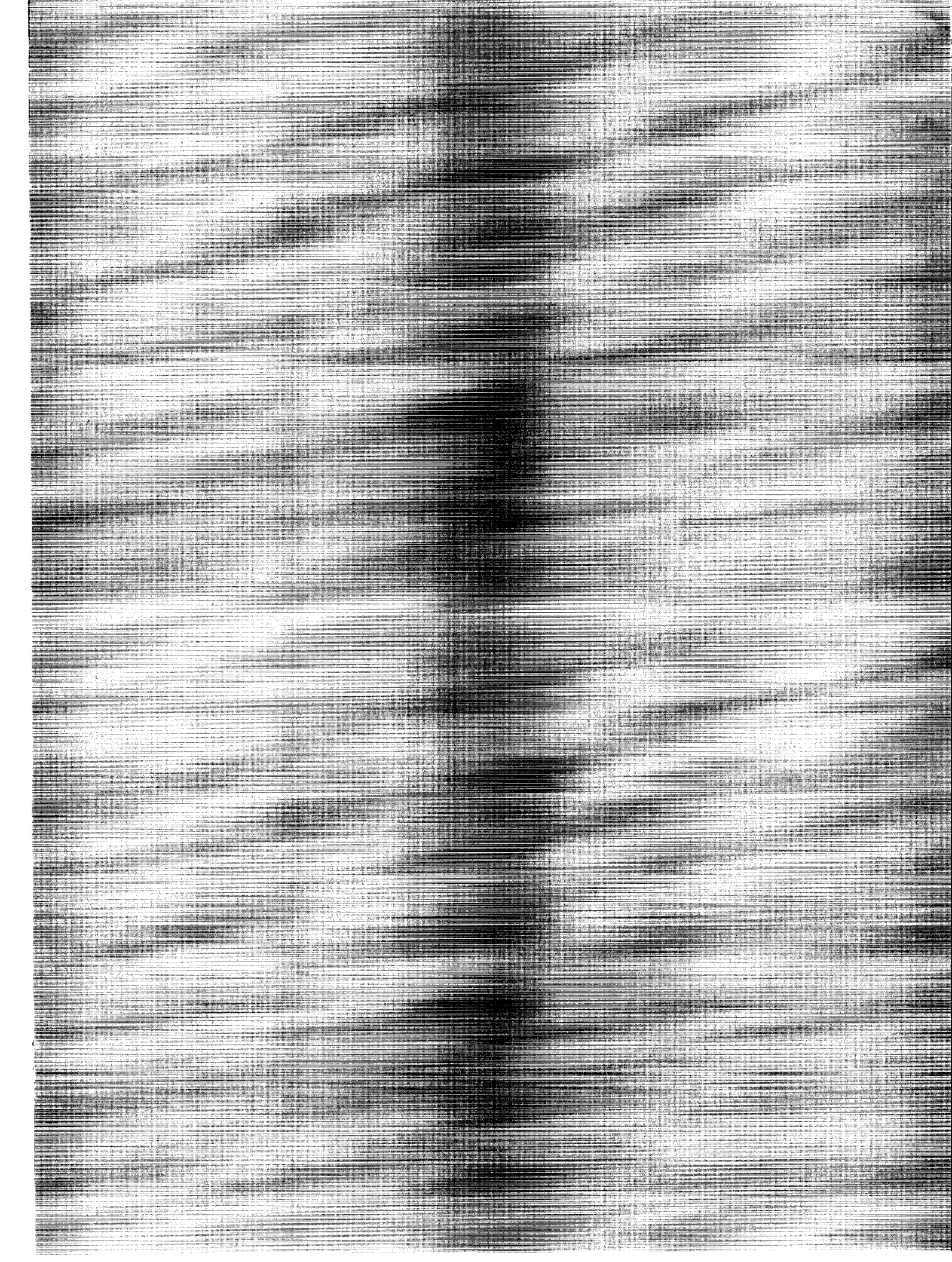
ISX plans to continue development of the the assistant system framework, our goal being the fielding of a commercial Manager's Assistant product currently code named "MAX."

We intend to pursue product development through both Government and private channels. Anticipated funding from SSFPO will support the development of the PAA, which will address the assembly sequence planning problem and should produce reusable concepts, designs, and possibly even code that can be applied to MAX. For the major portion of MAX development, however, we intend to seek a NASA Phase II SBIR to continue the current work and to seek funding from commercial sources that will leverage the NASA investment and increase the likelihood of success. We have already begun discussions with potential commercial investors, and we have found several potential sources of funding. To acquire this funding, however, we will need a fairly robust and full-featured prototype system that will help demonstrate the technical feasibility and market viability of MAX. We plan to use a Phase II SBIR to produce this system.

A Phase II SBIR would fund a one-year development effort that would produce the first MAX Infrastructure and Planner and apply it to a problem other than assembly sequence planning. The system would contain models (Problem Recognition, Problem Solving, and Resource Capability) for the application domain. Further development, to be supported by commercial sources, would produce models for other application domains and would further generalize the Infrastructure and Development Tools.

We believe that with NASA's continued support we can produce a successful commercial MAX product that will have broad applicability to professional problem solving in a variety of problem domains. .

©ISX Corporation



Application Assessment Report

Space Station Assembly Sequence Planning

Submitted By

ISX Corporation
501 Marin St., Suite 214
Thousand Oaks, CA 91360

March 6, 1990

Prepared For

William Bastedo
NASA Space Station Program Office
10701 Park Ridge Blvd.
Code SSS
Reston, VA 22091-4398

Table of Contents

Executive Summary

On February 26 and 27, 1990, ISX Corporation staff conducted an in-depth assessment of the Space Station Freedom (SSF) assembly sequence planning process. The assessment team included Bill Bewley, Gary Edwards, David Rosenberg, and Allen Smith of ISX. Peter Warren and Brook Sullivan of Booz-Allen & Hamilton were the application experts, and Bill Bastedo of the Space Station Freedom Program Office (SSFPO) provided application information in the form of feedback to a presentation of preliminary assessment results on the afternoon of February 27.

ISX application assessments are concentrated evaluations of potential applications for intelligent systems technology. The assessments are intended to confirm the preliminary determinations of the value and feasibility of an application and to develop an initial system design and implementation plan.

SSF assembly sequence planning is an integral and vital component of the ongoing SSF program. Its function, broadly stated, is to define launch vehicle flights and manifests that satisfy a complex, interdependent set of SSF, launch vehicle, and programmatic constraints. There are two phases of the process: the first is performed by SSFPO staff and involving development of the flight manifest (the inter-flight plan), definition and placement of cargo elements in the NSTS cargo bay (intra-flight plans), and documentation and publication of plans; and the second is performed by SSF engineering groups, which evaluate the documented plans, identify constraint violations, and provide feedback to SSFPO. There are two iteration loops in the process: internal iteration, with frequent iterations from plan validation by internal (SSFPO) analyses back to plan revision; and external iteration, with less frequent iterations from plan validation by external (SSF engineering) analyses to plan revision by SSFPO.

The assessment identified two major areas in which the assembly sequence planning process is problematic: the assembly sequence planner's work load and the maintenance of plan data. In both development of a new baseline plan and the performance of what-ifs, the planner is charged with developing the manifest, defining cargo elements, placing cargo elements in the NSTS cargo bay, calculating performance measures, validating the measures, identifying violated constraints, and revising invalidated parts of the plan. The analyses supporting these activities are time-consuming and must be performed under severe time pressure. The what-ifs are especially difficult in that time pressure is always great, they occur frequently, and the timing of their occurrence is difficult to predict. Because of the time pressure, planners find it difficult to perform the depth and breadth of analysis required to produce accurate and easily justifiable results.

Much of the assembly sequence planner's load is attributable to tedious and time-consuming analyses that involve executing procedures, applying well-known heuristics, and processing the horrendous detail of plan data, including assembly elements, constraints, and all their interdependencies. This is the intellectual "scut" work of analysis and planning, and the load it imposes often prevents the planner from spending time doing the creative problem solving required for assembly sequence planning. It also makes it difficult for the planner to provide quick responses to questions and to perform the depth and breadth of analysis needed to produce a new baseline plan or evaluate a what-if.

There is an opportunity to unload the planner by providing a personal "assistant" in a machine that would perform the work that is currently so tedious and time-consuming.

The assistant would be a ***personal analysis assistant*** that helps the human planner by doing the bookkeeping to maintain plan data and executing the procedures and heuristics currently used by the human planner to define flights, develop flight manifests, define and place cargo elements, calculate performance measures, and identify violated constraints. This unloading would speed the planning process, enable greater depth and breadth of analysis, and free the human planner to spend more time doing what only the human planner can do: evaluating analysis results; revising invalid assumptions, constraints and plans; generating new solutions to assembly sequence planning; and testing solution hypotheses with what-ifs.

The assessment indicated the value and feasibility of a Personal Analysis Assistant system to assist SSFPO and Booz-Allen & Hamilton staff in producing baseline assembly sequence plans and what-if analyses. Three major elements of such a system were identified: an Infrastructure for analysis support and validity checking; an Intra-Flight Value-Added function that would generate and place cargo elements given a manifest; and an Inter-Flight Value-Added function that would generate a manifest given major milestones.

A project plan was developed that supports delivery of the Infrastructure by the end of June, delivery of the Intra-Flight Value-Added function by the end of August, and delivery of the Inter-Flight Value-Added function by the end of September. Support tasks continue through the end of November.

Rough order of magnitude estimated burdened cost is presented in the table shown below. These estimates do not include the cost of SSFPO and Booz-Allen & Hamilton labor and travel. The estimate for the System Infrastructure deliverable is reduced by approximately $25K because Infrastructure development is partially supported by the NASA-Ames Phase 1 SBIR.

Deliverable	ISX Labor	ISX Travel	Total Cost
System Infrastructure	148.8	13.8	162.6
Intra-Flight Value-Added	127.1	9.9	137.0
Inter-Flight Value-Added	127.1	9.9	137.0
Support	86.7	2.6	89.3
Total	489.7	36.2	525.9

1.0 Introduction

On February 26 and 27, 1990, ISX Corporation staff conducted an in-depth assessment of the Space Station assembly sequence planning process. The assessment team included Bill Bewley, Gary Edwards, David Rosenberg, and Allen Smith of ISX. Peter Warren and Brook Sullivan of Booz-Allen & Hamilton were the application experts, and Bill Bastedo of SSFPO provided application information in the form of feedback to a presentation of preliminary assessment results on the afternoon of February 27.

The assessment was performed under a Phase 1 SBIR (Small Business Innovative Research) contract (Contract NAS2-13161) awarded to ISX by NASA's Ames Research Center to support the development of a knowledge-based decision-support system for Space Station engineering managers. Following meetings in November, 1989 with Paul Neumann and Ben Barker of the Space Station Freedom Program Office (SSFPO) and conversations with Henry Lum of NASA-Ames and Gregg Swietek of NASA-HQ, it was determined that Space Station Freedom (SSF) assembly sequence planning is an appropriate application for the Phase 1 SBIR and that the SSFPO is sufficiently interested to consider funding a task extension. ISX submitted a proposal for this possible task extension through NASA Research Announcement NRA2-34107(LMV). This submission, dated January 5, 1990, can be funded through March 31, 1990.

ISX application assessments are concentrated evaluations of potential applications for intelligent systems technology. The assessments are intended to confirm the preliminary determinations of the value and feasibility of an application and to develop an initial system design and implementation plan. The goals for the SSF assembly sequence planning application assessment were to:

- Analyze the assembly sequence planning process.
- Develop a preliminary system design and operational concept for an assembly sequence planning decision-support system.
- Assess the feasibility and value of developing a knowledge-based system to support the assembly sequence planning process.
- Develop a phased project plan for the design and implementation of an assembly sequence planning decision-support system that will provide required functionality.

The overall schedule for the February 26 - 27 assessment meetings, conducted at NASA Space Station Program Office facility in Reston, VA, was as follows:

Monday, 2/26/90:

- introduction and discussion of assessment objectives
- an overview of the nature of assembly sequence planning
- presentation of planning constraints associated with the NSTS, the SSF configuration, and external commitments
- presentation, analysis, and discussion of assembly sequence planning cases

- characterization of roles to be played by a knowledge-based assembly sequence planning decision-support system

Tuesday, 2/27/90:

- ISX presentation to Peter Warren and Brook Sullivan summarizing the application overview, assessment, and a preliminary design for a Personal Analysis Assistant for assembly sequence planning

- ISX presentation to Bill Bastedo, revised following discussion with Peter and Brook, summarizing the application overview, assessment, and a preliminary design for a Personal Analysis Assistant

The main body of this report is presented in six sections:

- Section 1 comprises this Introduction

- Section 2 presents an analysis of the assembly sequence planning process as described by Mr. Warren, Mr. Sullivan, and Mr. Bastedo. The analysis includes an overview of the process, problems in the process, and the need for a knowledge-based decision-support system.

- Section 3 presents a preliminary design for a Personal Analysis Assistant, a knowledge-based decision-support system for assembly sequence planning.

- Section 4 analyzes the proposed system in terms of value and feasibility.

- Section 5 presents a project plan for the development of the Personal Analysis Assistant.

- Section 6 summarizes estimated costs associated with Personal Analysis Assistant development.

2.0 Analysis of the Assembly Sequence Planning Process

2.1 Overview of the Process

Figure 1 presents a graphic overview of the SSF assembly sequence planning process. There are two phases of the process: the first is performed by SSFPO staff and involving development of the flight manifest (the inter-flight plan), definition and placement of cargo elements in the NSTS cargo bay (intra-flight plans), and documentation and publication of plans; and the second is performed by SSF engineering groups, which evaluate the documented plans, identify constraint violations, and provide feedback to SSFPO. There are two iteration loops in the process: internal iteration, with frequent iterations from plan validation by internal (SSFPO) analyses back to plan revision; and external iteration, with less frequent iterations from plan validation by external (SSF engineering) analyses to plan revision by SSFPO.

The process begins with definition of constraints in three categories: NSTS constraints, e.g., volume and mass capacity, flight rate; SSF hardware constraints, e.g., assembly elements; and programmatic constraints, e.g., major milestones.

Given a space station configuration description (complete with definitions for all assembly elements) and major milestone constraints, e.g., dates and capability requirements for First Element Launch (FEL), Man-Tended Capability (MTC), Permanent Manned Capability (PMC), and Assembly Complete (AC), the SSFPO assembly sequence planner develops the flight manifest which consists of determining the number and timing of flights and assigning assembly elements to flights. Following definition of the manifest, intra-flight planning defines cargo elements as compositions of assembly elements manifested to the flight and places cargo elements in the cargo bay.

The manifest and its associated intra-flight cargo-element groupings and cargo bay placements are in essence a "hypothesis" which is tested by comparing performance measures derived from the hypothesis with standards defined by programmatic/milestone constraints, NSTS constraints, and SSF hardware constraints. Measures include mass, volume, center of gravity (CG), the power requirement, the intra- and extra-vehicular activity (EVA and IVA) requirements, and the Remote Manipulator System (RMS) reach requirement. If measures and/or margins violate constraints, the invalidated part of the plan is revised by changing/relaxing constraints or changing elements of the plan: the manifest, cargo element definition, or cargo bay placement.

When the SSFPO is satisfied with the validity of plans (normally after informal consultation with external groups), they are documented in a Space Station Stage Summary Databook, which is distributed for review to SSF engineering groups, who perform analyses and identify constraint violations. SSFPO collects feedback on constraint violations and uses the feedback to drive plan revisions.

Although this description may suggest that assembly sequence planning begins anew with each planning cycle, with constraints and plans being defined and developed from scratch on each iteration, assembly sequence planning is in fact an ongoing process in which constraints and plans are revisions of prior constraint definitions and plans.

There are two operational scenarios within the ongoing planning process: the first is development of a new baseline assembly sequence plan for documentation in the Space Station Stage Summary Databook; the second is performing what-if analyses that provide

responses to questions on proposed perturbations of the current baseline, e.g., "What if the NSTS mass capacity was increased by 12,000 lbs?".

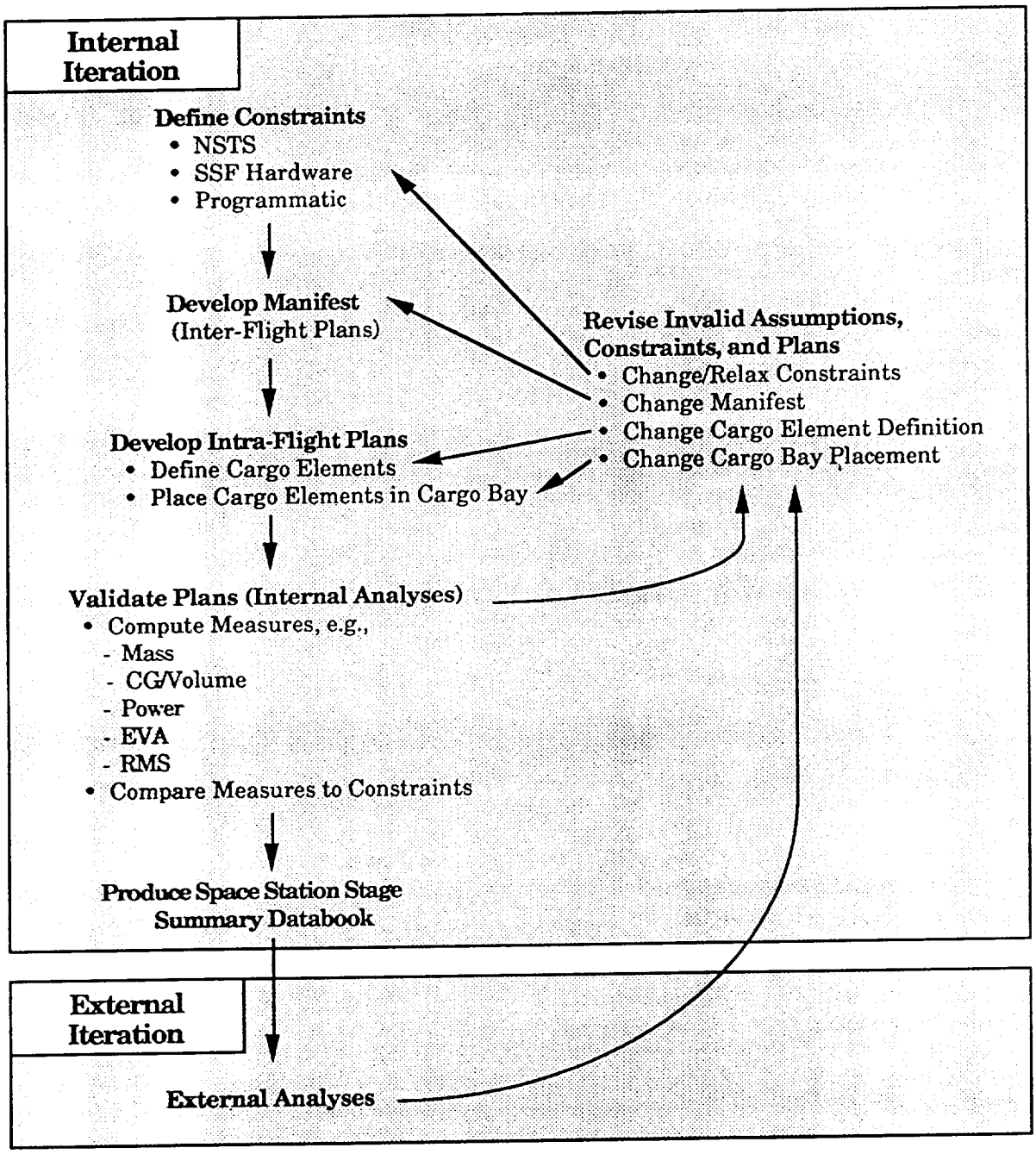


Figure 1: The SSF Assembly Sequence Planning Process

2.2 Problems with the Process

The assessment identified two major areas in which the assembly sequence planning process is problematic; these are the assembly sequence planner's work load, the maintenance of plan data, and communication of assembly sequence plans.

The Planner's Load. In both development of a new baseline plan and the performance of what-ifs, the planner is charged with developing the manifest, defining cargo elements, placing cargo elements in the NSTS cargo bay, calculating performance measures, validating the measures, identifying violated constraints, and revising invalidated parts of the plan. The analyses supporting these activities are time-consuming and must be performed under severe time pressure. The what-ifs are especially difficult in that time pressure is always great, they occur frequently, and the timing of their occurrence is difficult to predict. Because of the time pressure, planners find it difficult to perform the depth and breadth of analysis required to produce accurate and easily justifiable results.

Maintenance of Plan Data. It is a gross understatement to say that assembly sequence planning is complex. There are 300 assembly elements and hundreds of constraints, all of which interact. Insuring that all assembly elements are manifested, associating assembly operational flow and data on mass, CG, volume, power, EVA, RMS, and heat loads with each manifest, and keeping track of constraints that have been and have not been met is a huge bookkeeping task that is both time-consuming and prone to error.

Communication of Assembly Sequence Plans. The communication of complex assembly sequence plans to the broader space station community is a difficult problem in its own right. A large number of Government, contractor, and subcontractor organizations need ready access to the assembly sequence and its supporting data. Since many organizations external to the SSFPO perform extensive and detailed evaluations of a assembly sequence, some reliable and timely feedback mechanism would be most desirable.

2.3 The Need and the Opportunity

Much of the assembly sequence planner's load is attributable to tedious and time-consuming analyses that involve executing procedures, applying well-known heuristics, and processing the horrendous detail of plan data, including assembly elements, constraints, and all their interdependencies. This is the intellectual "scut" work of analysis and planning, and the load it imposes often prevents the planner from spending time doing the creative problem solving required for assembly sequence planning. It also makes it difficult for the planner to provide quick responses to questions and to perform the depth and breadth of analysis needed to produce a new baseline plan or evaluate a what-if.

Since computers are better and faster at this intellectual scut work than humans, there is an opportunity to unload the planner by providing a personal "assistant" in a machine that would perform the work that is currently so tedious and time-consuming. The assistant would be a ***personal analysis assistant*** that helps the human planner by doing the bookkeeping to maintain plan data and executing the procedures and heuristics currently used by the human planner to define flights, develop flight manifests, define and place cargo elements, calculate performance measures, and identify violated constraints. This unloading would speed the planning process, enable greater depth and breadth of analysis, and free the human planner to spend more time doing what only the human planner can do: evaluating analysis results; revising invalid assumptions, constraints and plans; generating new solutions to assembly sequence planning; and testing solution hypotheses with what-ifs.

3.0 Preliminary Design for a Personal Analysis Assistant

The Personal Analysis Assistant is designed to provide the assistance needed to unload the assembly sequence planner. It is an intelligent system that combines knowledge-based and conventional algorithmic technologies to produce an integrated decision-support system for the assembly sequence planner. As shown in Figure 2, the design is evolutionary in that it is based on a framework or infrastructure that provides value to the planner at the earliest stages of development and builds on that value by supporting the addition of new functionality and the enhancement of existing functionality over the life of the system.

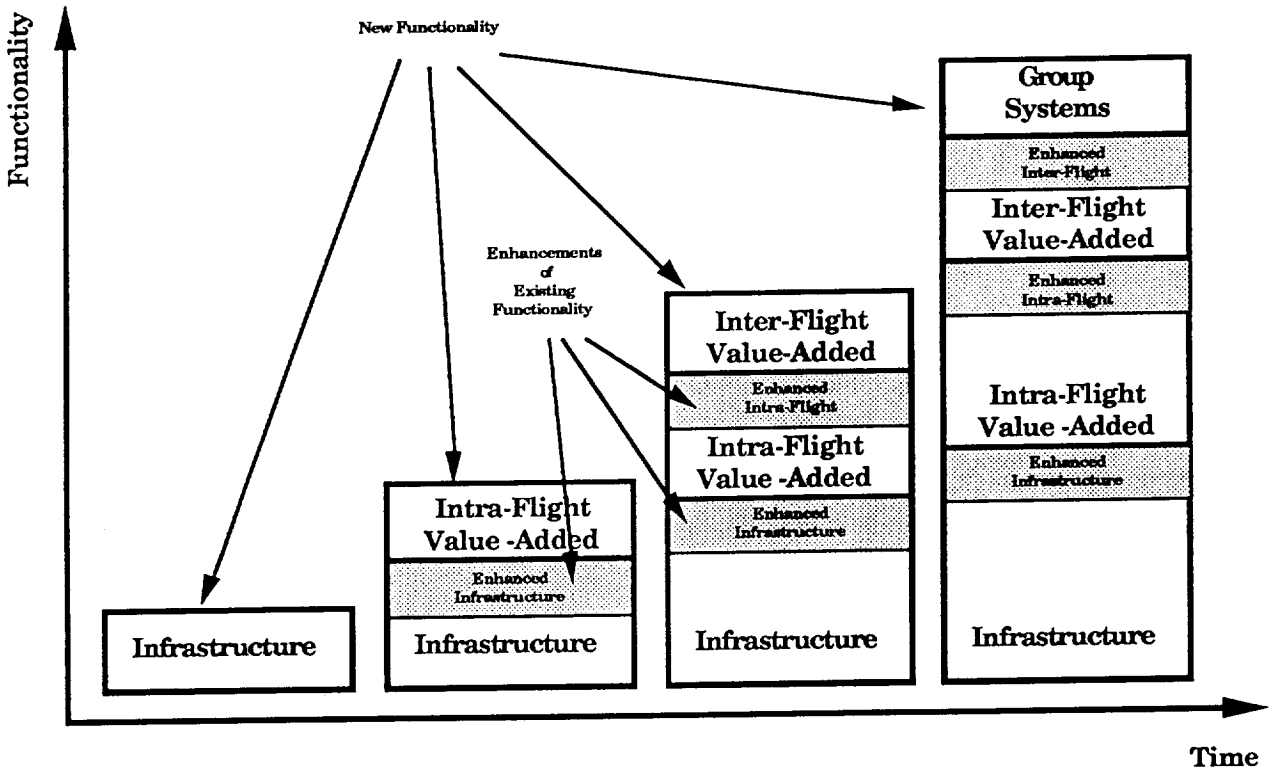


Figure 2: The Design is Evolutionary, Providing Early Value and Supporting the Addition and Enhancement of Functionality.

Assembly sequence planning is a difficult problem for humans, and full automation is beyond the capability of current technology (and probably not desirable, in any case). The system is, therefore, designed to assist the human planner, not to automate the assembly sequence planning process. The planner will specify the tasks the system is to perform, e.g., validate specified flight plans, define cargo elements and place them in the cargo bay of a flight, or define flights and flight manifests for specified major milestones. The planner will also specify the assumptions the system is to use, e.g., flight plans, assembly elements, manifests, and constraints. Given these specifications, the system will generate results. The planner will then review the results, possibly request explanations of results or follow-up analyses, and then make the final decisions, potentially modifying the computer-generated plan. The system is essentially a laboratory for the assembly sequence planner, providing an environment in which the planner can define and run

conceptual experiments that help him do the problem solving needed to generate plans.

3.1 System Requirements

The assessment identified the following requirements that must be met by a Personal Analysis Assistant for SSFPO assembly sequence planners:

- Provide an analysis support infrastructure for defining constraints, the manifest, cargo elements, placement of cargo elements in the NSTS cargo bay, functions that propagate constraint changes through the network of constraints, and an interface for user input and the specification and generation of output reports.

- Provide functions for calculating intra-flight performance measures, e.g., mass, CG, and volume and validating intra-flight plans by comparing measures to constraints.

- Provide functions for defining cargo elements and planning placement of cargo elements.

- Provide functions for validating inter-flight plans by comparison to constraints.
- Provide functions for developing flight manifests from given programmatic constraints.

- Support extensibility of the Personal Analysis Assistant by SSFPO and Booz-Allen & Hamilton staff.

- Run on Macintosh II series platforms widely available at the SSFPO and Booz-Allen & Hamilton facilities.

3.2 Functional Architecture

Figure 3 shows the functional architecture of the Personal Analysis System. The shaded box, External Interfaces, and the dotted lines represent functionality that is supported by the design but is outside the scope of the proposed work.

The functions shown in the figure are grouped into three major software elements: the System Infrastructure, Intra-Flight Value-Added, and Inter-Flight Value-Added. These software elements are described below.

The **System Infrastructure** will provide the following functions:

- the user interface, which includes an object editor for defining assembly sequence plan elements (e.g., flights, assembly elements, cargo elements, and manifests) as well as constraints

- the Net Builder, which will generate the network of constraints, plan elements, and their relations

- the Net Manager, which will update the network by propagating constraint changes

- Value Calculators, which will compute values of performance measures, e.g., mass, volume, center of gravity (CG), the power requirement, the extra-vehicular activity (EVA) requirement, and Remote Manipulator System (RMS) reach

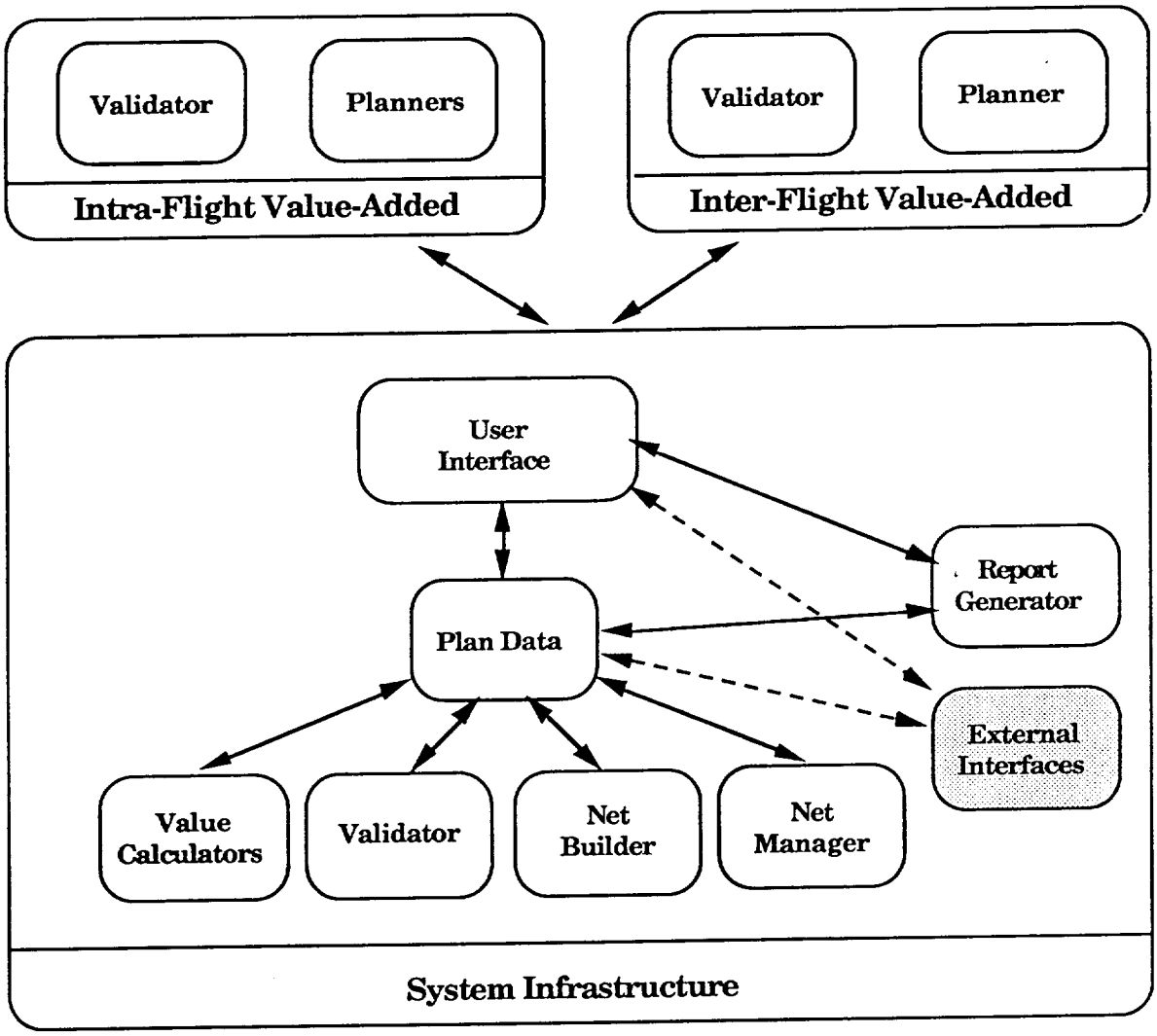


Figure 3: Functional Architecture of the Personal Analysis Assistant

- a Validator, which will compare values of performance measures to standards and determine whether measures violate constraints

The System Infrastructure provides the capability for building representations of assembly sequence plans, e.g., flights, manifests, cargo elements, and associated performance measures; networks of dependencies among plan objects; constraint networks describing bounds on measures; and user-specifications of constraints and analysis options. It also supports calculation of performance measures and identification of violated constraints, and intra-flight plans. This corresponds to "Validate Plans" in Figure 1. Finally, it is the substrate on which functions providing inter-flight and intra-flight value-added functionality are built.

Intra-Flight Value-Added will provide the following functions:

- an enhancement to the user interface in the form of editor for defining intra-flight constraints, dependencies, and abstractions
- an intra-flight Validator, which will determine whether cargo element definitions or cargo element placements violate constraints
- Planners, which will define cargo elements and placement of cargo element in the NSTS cargo bay

Intra-Flight Value-Added will provide functionality that will, given the manifest for each flight, define cargo elements, place them in the NSTS cargo bay, compute performance measure values, and identify violated constraints. This corresponds to support of "Develop Intra-Flight Plans" and "Validate Plans" in Figure 1.

Inter-Flight Value-Added, which will provide:

- an enhancement to the user interface in the form of editor for defining inter-flight constraints, dependencies, and abstractions
- an inter-flight Validator, which will determine whether the manifest violates constraints
- a Planner, which will define flights and manifest assembly elements to flights given a space station configuration, program milestones, NSTS description, etc.

Inter-Flight Value-Added will provide functionality that will, given major milestone dates and capability requirements, determine the number and dates of required flights and develop a manifest for each flight.

3.3 Operational Scenarios

The proposed system will support the following operational scenarios:

- Given the manifest and definition and placement of cargo elements, the system validates intra- and inter-flight plans by computing performance measures, comparing measures to constraints, and flagging violated constraints. Enabling Personal Analysis Assistant functions are: System Infrastructure and Intra-Flight Value-Added.
- Given the manifest, the system defines and places cargo elements, and then validates intra- and inter-flight plans by computing performance measures, comparing measures to constraints, and flagging violated constraints. Enabling Personal Analysis Assistant functions are: System Infrastructure and Intra-Flight Value-Added.
- Given major milestones, the system defines the number and dates of required flights and develops a manifest for each flight, then defines and places cargo elements, and finally validates intra- and inter-flight plans by computing performance measures, comparing measures to constraints, and flagging violated constraints. Enabling Personal Analysis Assistant functions are: System

Infrastructure, Intra-Flight Value-Added, and Inter-Flight Value-Added.

- Given a request for a what-if based on a constraint change, the system propagates the change, identifies plan elements affected by the change, develops new plans for the affected elements, and then validates changed plans by computing performance measures, comparing measures to constraints, and flagging violated constraints. Enabling Personal Analysis Assistant functions are: System Infrastructure, Intra-Flight Value-Added, and Inter-Flight Value-Added.

- Given a request for a what-if based on a plan change by the user, the system propagates the change, develops new plans for the affected elements, and then validates changed plans by computing performance measures, comparing measures to constraints, and flagging violated constraints. Enabling Personal Analysis Assistant functions are: System Infrastructure, Intra-Flight Value-Added, and Inter-Flight Value-Added.

4.0 The Feasibility and Value of a Personal Analysis Assistant

The assessment employed a formal methodology developed by ISX over several years and hundreds of projects. Based on case-based interviews of experts and users, the methodology rates the proposed application on scales measuring aspects of AI Risk (the application risk associated with the artificial intelligence aspects of the proposed system), Systems Engineering Risk (the risk associated with combining intelligent modules with conventional modules to build a large and complex intelligent system), the value provided by the application, and the estimated cost of the application.

The process begins with begins with a presentation by the expert summarizing the following information:

- An application overview, including:
  - Characterization of the application
  - Types of application situations
  - The overall flow of steps or decisions in performing the application
  - Expertise used in performing the application
  - Opportunities for improving on the current practice

- An Example Case, including:
  - Circumstances surrounding the case
  - Step by step description of the actions that were taken in handling the case
  - The expert's analysis of how the case was handled

From this presentation, the assessment team acquired information needed for completion of the "Application Screening Profile," a tool developed by ISX staff for use in assessing potential applications.

4.1 The Application Screening Profile

The Application Screening Profile, which is shown in Figure 4, characterizes a potential application by scaling it on three dimensions: AI Risk, Systems Engineering Risk, and Value & Cost. Appendix A provides complete definitions for this rating system.

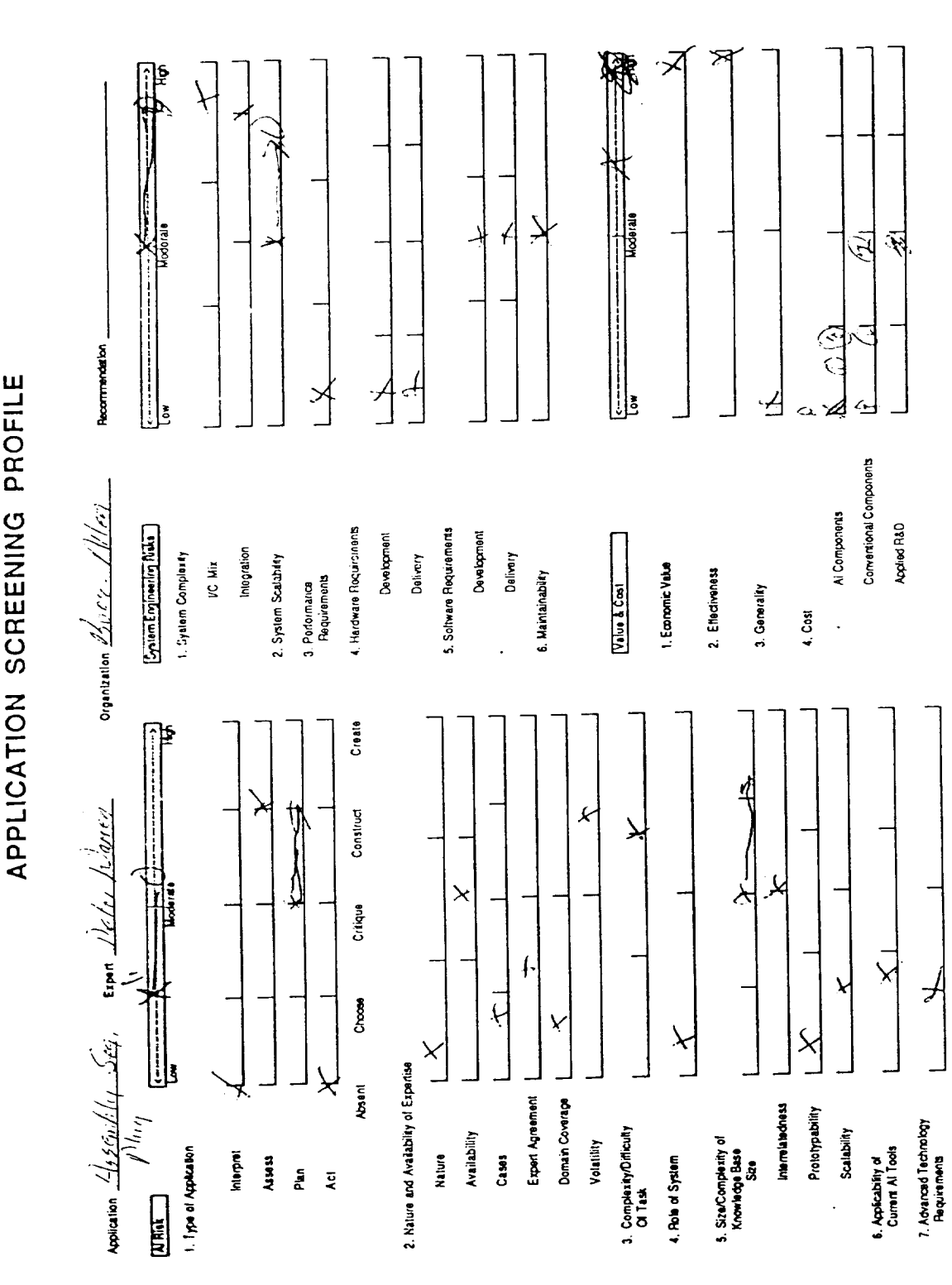


Figure 4: The Application Screening Profile, With Personal Analysis Assistant Ratings

4.2 Assessment Results

Ratings of the Personal Analysis Assistant are shown on Figure 4. The rationale for the ratings is summarized below:

- AI Risk -- low/moderate risk. This rating is based on the ready availability of expertise, the anticipated moderate size of the knowledge base, the applicability of current AI tools, and the assessment/planning nature of the application, for which a variety of proven AI techniques are available.

- Systems Engineering Risk -- moderate/high risk. Risk is elevated by strong requirements for scaleability and integration of intelligent and conventional components. This risk is mitigated by the system design and the development plan, which supports system extensibility by SSFPO and Booz-Allen & Hamilton staff. Hardware and software development/delivery requirements are not difficult to achieve.

- System Value -- high. The system is expected to enhance assembly sequence planning by supporting more timely, less costly, and more effective planning and what-if analyses. Assembly sequence planning is an extremely high-value task, giving the resulting system good leverage.

- System Cost -- low/moderate. Cost is relative, of course, but development of AI components are expected to require approximately 1 person-year, which is low compared to operational expert systems. Systems engineering cost is estimated to be approximately 5 person-years. Estimated rough order of magnitude cost is presented in Section 6.0.

5.0 Project Plan

Based on several years of experience developing systems such as Pilot's Associate, AIM, and ALSYM, ISX has developed an Intelligent Systems Engineering (ISE) methodology to meet the unique methodological requirements imposed by systems that combine knowledge-based components with conventional software components. The ISE methodology, which blends the techniques of conventional systems engineering with those of knowledge engineering and rapid prototyping, will be applied to the development of the Personal Analysis Assistant.

Like conventional systems engineering methodologies, ISE proceeds through requirements analysis, design, and implementation phases. Unlike conventional methodologies, however, the ISE methodology employs rapid prototypes throughout the development process, beginning with requirements analysis, to support user-centered definition of requirements and system design and the acquisition and engineering of the expert knowledge on which intelligent systems are based.

In systems with strong extensibility requirements, such as the Personal Analysis Assistant, the methodology also involves user engineers as members of the development staff to insure extensibility and maintainability of the system by the user organization. This planned "technology transfer" is a primary feature of the ISE methodology.

The project plan described in this section is based on the ISE methodology and therefore assumes significant involvement by experts and an engineer supplied by the user organization. The estimated level of this involvement is listed for each task described in Section 5.3 and is summarized in the description of the team presented in 5.4.

5.1 Deliverables

There are four major deliverables, the first three corresponding to the major system elements described in Section 3 and the fourth being system documentation. Rapid prototypes will be produced in support of each of the three software deliverables.

System Infrastructure
This will provide the functionality described in Section 3.2. It will include the user interface, an object editor for defining assembly sequence plan elements, constraints, dependencies, and abstractions, the Net Builder, and the Net Manager. It will provide the capability for building representations of assembly sequence plans, e.g., flights, manifests, cargo elements, and associated performance measures; networks of dependencies among plan objects; constraint networks describing bounds on measures; and user-specifications of constraints and analysis options. It also supports calculation of performance measures and identification of violated constraints, and intra-flight plans, and it is the substrate on which inter-flight and intra-flight value-added functionality elements are built. The user interface will support all user input and will provide functionality supporting user specification of on-screen and printed report formats.

Intra-Flight Value-Added
As described in Section 3.2, this will include an editor for defining intra-flight constraints, dependencies, and abstractions, a Validator, and Planners that will define cargo elements and placement of cargo element in the NSTS cargo bay. It will provide the capability for the system to define cargo elements from a given manifest, place cargo elements in the cargo bay, compute performance measures, and identify violated

constraints.

Inter-Flight Value-Added
Also described in Section 3.2, this will include an editor for defining inter-flight constraints, dependencies, and abstractions, a Validator, and a Planner, which will define flight dates and manifest assembly elements to flights. It will provide the capability to determine the number and dates of required flights and develop a manifest for each flight, given major milestone dates and capability requirements.

Documentation
A User's Manual will be provided that contains information required to support use in performing assembly sequence planning and extension of the system to support addition of constraints and assembly sequence objects and their relations.

5.2 Schedule and Milestones

The assumed development period is 38 weeks, with the period of performance running from March 12, 1990 to November 30, 1990. Our on-going SBIR enables the early start. The schedule and milestones are shown in Figure 5, and tasks are described in Section 5.3.

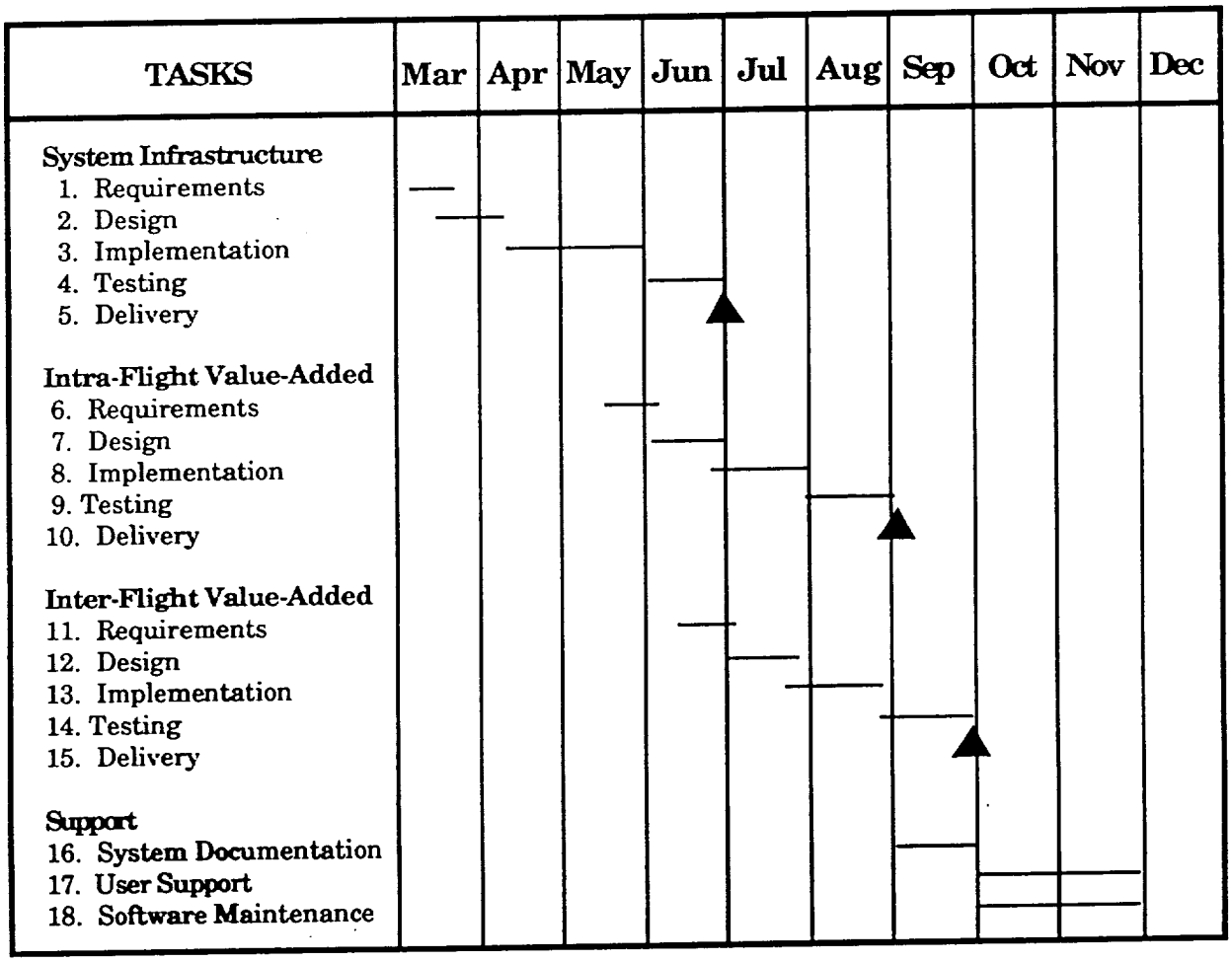


Figure 5: Schedule and Milestones

5.3 Task Descriptions

Task 1: System Requirements Analysis
Purpose: Define Personal Analysis Assistant system requirements.

Conduct: Meetings involving ISX knowledge engineers and SSFPO/Booz-Allen & Hamilton experts/users. Review cases and develop operational scenarios. Generate object lists, data dictionary. Describe system states, user information requirements, and system responses.

Duration: 2 weeks.

Level of Expert Involvement: 2 days.

Level of Booz-Allen & Hamilton Technical Involvement: 2 days.

Task 2: System Design
Purpose: Produce the overall design for the Personal Analysis Assistant and a detailed design for the System Infrastructure.

Conduct: Review preliminary design of system and collect information on all current and anticipated constraints and assembly sequence planning objects that will support design of the Net Builder, the Net Manager and the User Interface. ISX designer characterizes operational scenarios developed in Task 5, identifying major tasks and task groupings. Identifies major subfunctions, performs technology assessment, specifies major system components and their interactions, assigns subsystem functionality, and decomposes scenario events by subsystem.

Duration: 4 weeks.

Level of Expert Involvement: 1 day.

Level of Booz-Allen & Hamilton Technical Involvement: 4 days.

Task 3: Infrastructure Implementation
Purpose: Build an initial version of the Infrastructure.

Conduct: Implement the Infrastructure through a series of rapid prototypes, proceeding in a depth-first fashion with SSFPO/Booz-Allen & Hamilton experts/users reviewing and providing feedback on each prototype.

Duration: 6 weeks.

Level of Expert Involvement: 2 days.

Level of Booz-Allen & Hamilton Technical Involvement: 8 days.

Task 4: Infrastructure Testing
Purpose: Iterative improvement of the Infrastructure's capabilities through case-based testing and refinement of functionality.

Conduct: Extensively exercise the system using expert-supplied cases, using test results to improve the system. The expert plays a key role, providing the knowledge engineer's with additional information about constraints, assembly sequence objects, and user interface preferences.

Duration: 4 weeks.

Level of Expert Involvement: 2 days.

Level of Booz-Allen & Hamilton Technical Involvement: 5 days.

Task 5: Deliver the Infrastructure
Purpose: Milestone for delivery of the Infrastructure.

Conduct: Demonstration and code delivery to SSFPO/Booz-Allen & Hamilton experts/users and NASA-Ames.

Duration: 1 day.

Level of Expert Involvement: 1 day.

Level of Booz-Allen & Hamilton Technical Involvement: 1 day.

Task 6: Intra-Flight Value-Added Requirements Analysis
Purpose: Define Intra-Flight Value-Added requirements.

Conduct: Meetings involving ISX knowledge engineers and SSFPO/Booz-Allen & Hamilton experts/users. Review cases and develop operational scenarios. Generate object lists, data dictionary. Describe subsystem states, user information requirements, and system responses.

Duration: 2 weeks.

Level of Expert Involvement: 2 days.

Level of Booz-Allen & Hamilton Technical Involvement: 2 days.

Task 7: Intra-Flight Value-Added Design
Purpose: Produce the Intra-Flight Value-Added design.

Conduct: ISX designer characterizes operational scenarios developed in Task 10, identifying major tasks and task groupings. Identifies major subfunctions, performs technology assessment, specifies major system components and their interactions, assigns subsystem functionality, and decomposes scenario events by subsystem.

Duration: 4 weeks.

Level of Expert Involvement: 1 day.

Level of Booz-Allen & Hamilton Technical Involvement: 4 days.

Task 8: Intra-Flight Value-Added Implementation
Purpose: Build an initial version of the Intra-Flight Value-Added system.

Conduct: Implement the Infrastructure through a series of rapid prototypes, proceeding in a depth-first fashion with SSFPO/Booz-Allen & Hamilton experts/users reviewing and providing feedback on each prototype.

Duration: 5 weeks.

Level of Expert Involvement: 1 day.

Level of Booz-Allen & Hamilton Technical Involvement: 8 days.

Task 9: Intra-Flight Value-Added Testing
Purpose: Iterative improvement of the Intra-Flight Value-Added system's capabilities through case-based testing and refinement of functionality.

Conduct: Extensively exercise the system using expert-supplied cases, using test results to improve the system. The expert plays a key role, providing the knowledge engineer's with additional information about constraints, assembly sequence objects, and user interface preferences.

Duration: 4 weeks.

Level of Expert Involvement: 2 days.

Level of Booz-Allen & Hamilton Technical Involvement: 5 days.

Task 10: Deliver Intra-Flight Value-Added
Purpose: Milestone for delivery of Intra-Flight Value-Added.

Conduct: Demonstration and code delivery to SSFPO/Booz-Allen & Hamilton experts/users and NASA-Ames.

Duration: 1 day.

Level of Expert Involvement: 1 day.

Level of Booz-Allen & Hamilton Technical Involvement: 1 day.

Task 11: Inter-Flight Value-Added Requirements Analysis
Purpose: Define Inter-Flight Value-Added requirements.

Conduct: Meetings involving ISX knowledge engineers and SSFPO/Booz-Allen & Hamilton experts/users. Review cases and develop operational scenarios. Generate object lists, data dictionary. Describe subsystem states, user information requirements, and system responses.

Duration: 2 weeks.

Level of Expert Involvement: 1 day.

Level of Booz-Allen & Hamilton Technical Involvement: 2 days.

Task 12: Inter-Flight Value-Added Design
Purpose: Produce the design for the Inter-Flight Value-Added system.

Conduct: ISX designer characterizes operational scenarios developed in Task 16, identifying major tasks and task groupings. Identifies major subfunctions, performs technology assessment, specifies major system components and their interactions, assigns subsystem functionality, and decomposes scenario events by subsystem.

Duration: 4 weeks.

Level of Expert Involvement: 1 day.

Level of Booz-Allen & Hamilton Technical Involvement: 4 days.

Task 13: Inter-Flight Value-Added Implementation
Purpose: Build an initial version of the Intra-Flight Value-Added system.

Conduct: Implement the Inter-Flight Value-Added system through a series of rapid prototypes, proceeding in a depth-first fashion with SSFPO/Booz-Allen & Hamilton experts/users reviewing and providing feedback on each prototype.

Duration: 5 weeks.

Level of Expert Involvement: 1 day.

Level of Booz-Allen & Hamilton Technical Involvement: 8 days.

Task 14: Inter-Flight Value-Added Testing
Purpose: Iterative improvement of the Inter-Flight Value-Added system's capabilities through case-based testing and refinement of functionality.

Conduct: Extensively exercise the system using expert-supplied cases, using test results to improve the system. The expert plays a key role, providing the knowledge engineer's with additional information about constraints, assembly sequence objects, and user interface preferences.

Duration: 4 weeks.

Level of Expert Involvement: 2 days.

Level of Booz-Allen & Hamilton Technical Involvement: 5 days.

Task 15: Deliver Inter-Flight Value-Added
Purpose: Milestone for delivery of Inter-Flight Value-Added.

Conduct: Demonstration and code delivery to SSFPO/Booz-Allen & Hamilton experts/users and NASA-Ames.

Duration: 1 day.

Level of Expert Involvement: 1 day.

Level of Booz-Allen & Hamilton Technical Involvement: 1 day.

Task 16: System Documentation
Purpose: Write the User's Manual.

Conduct: ISX and Booz-Allen & Hamilton engineers write documentation containing information required to perform both assembly sequence planning and extension of the system to support addition of constraints and assembly sequence objects and their relations.

Duration: 4 weeks.

Level of Expert Involvement: 0 days.

Level of Booz-Allen & Hamilton Technical Involvement: 5 days.

Task 17: User Support
Purpose: Provide support to SSFPO and Booz-Allen & Hamilton users.

Conduct: ISX engineers provide telephone support and on-site support (1 on-site visit) to SSFPO and Booz-Allen & Hamilton users. Support is in the nature of assistance in using the system as delivered.

Duration: 8 weeks.

Task 18: Software Maintenance
Purpose: Provide software maintenance support.

Conduct: ISX engineers correct software problems discovered during post-delivery us of the Personal Analysis Assistant.

Duration: 8 weeks.

5.4 The Development and Management Team

The project team for the development of the Personal Analysis Assistant requires a blend of several types of experience and expertise. Development and management roles are described below, along with estimated level of effort for each role.

Primary Expert
Experience has proven that for most effective conduct of the project there should be a single operational specialist available for frequent extended interactions with the development team. The expert will:

- provide application expertise and cases for design, development, and testing purposes
- serve as the prime critiquer of the emerging Personal Analysis Assistant system

The estimated level of effort for expert involvement (both primary and secondary) is 1 person-month.

Secondary Expert(s)
The person(s) in this role serve in a consulting capacity to the primary expert. The primary expert is responsible for the duties listed above, but may use the secondary expert(s) as a "sounding board," resource, and so on.

Booz-Allen & Hamilton Engineer
Extensibility is a major requirement for the Personal Analysis Assistant. To insure that the system will indeed be extensible by the user organization, a Booz-Allen & Hamilton engineer must be part of the development team. This person will be responsible for developing selected elements of the system. The estimated level of effort for this role is 3 person-months.

ISX Technical Lead
This person has technical responsibility for the design and development of the Personal Analysis Assistant. This includes both performance of the majority of design tasks and technical supervision of ISX implementors. Gary Edwards is proposed as the ISX Technical Lead. The estimated level of effort for this role is 6 person-months.

ISX Design Consultant
The person in this role will provide technical advice to the ISX Technical Lead and the ISX Project Lead. This assistance will include: design reviews; participation in technology assessment; exploration, evaluation, and recommendation on interfaces to external packages and databases; and exploration, evaluation, and recommendation on internal database technology and interfaces. David Rosenberg is proposed as the ISX Design Consultant. The estimated level of effort for this role is 3 person-months.

ISX Implementors
The persons in this role will implement and test elements of the Personal Analysis Assistant. They will be assisted by the Booz-Allen & Hamilton engineer. The estimated level of effort for this role is 14 person-months.

ISX Project Lead
The ISX Project Lead will have overall responsibility for delivery of the Personal Analysis Assistant. He will monitor and direct all work at the program level, and he will be the program-level contact to the SSFPO. He will also participate in requirements analysis and design. Bill Bewley is proposed as the ISX Project Lead. The estimated level of effort for this role is 6 person-months.

6.0 Estimated Cost

Rough order of magnitude estimated burdened cost is presented in Table 1. These estimates do not include the cost of SSFPO and Booz-Allen & Hamilton labor and travel. The estimate for the System Infrastructure deliverable is reduced by approximately $25K because Infrastructure development is partially supported by the NASA-Ames Phase 1 SBIR.

The cost of each deliverable is presented separately to enable selection of options, e.g., the Infrastructure, the Infrastructure and Intra-Flight Value-Added, and so on.

Table 1: Cost Estimates ($K)

Deliverable	ISX Labor	ISX Travel	Total Cost
System Infrastructure	148.8	13.8	162.6
Intra-Flight Value-Added	127.1	9.9	137.0
Inter-Flight Value-Added	127.1	9.9	137.0
Support	86.7	2.6	89.3
Total	489.7	36.2	525.9

Appendix A

ISX Application

Assessment Methodology

This appendix outlines the ISX application assessment methodology, defining the terms used in Figure 4.

AI Risk

AI Risk is the application risk associated with the artificial intelligence aspects of the proposed system. Subscales measure amounts of qualities that past experience in developing AI applications indicates increases the difficulty of development or reduces the likelihood of success.

1. Type of Application.

This subscale is actually composed of two subscales: the activities present in the application and the type of cognitive processing occurring within each activity.

There are four potential phases of activity within any application:

- Interpret. Transformation of signals from the external environment to a symbolic representation. Examples: Seismic Signals to Wave Form Phase Data; Optical Signals to Intensity/Time Ratio.
- Assess. Receive intermediate classifications as input and produce a higher-level classification (situation assessment) for use by Planning. Examples: Wave Form Phase Data to Nuclear Explosion; Diagnostic Test Results to Implied Faults.
- Plan. Receive a situation assessment as input and produce a plan as output. Examples: Implied Faults to Repair; Bogey to Tactics Plan.
- Act. Execute a plan. This phase occurs only in autonomous systems.

Within each activity phase, there are four possible types of cognitive processing that can occur. More than one of the following processing types can occur in any phase. In general, difficulty and implementation risk increases from Choose to Critique to Construct to Create.

- Choose. Select from among a fixed, enumerable set of alternatives.
- Critique. Analyze and evaluate a composite entity relative to a fixed standard, using criteria for correctness and completeness.
- Construct. Assemble a composite entity from a fixed, enumerable set of constituent element types, subject to a fixed set of rules.
- Create. Build a composite entity from a set of constituent element types, subject to rather general principles or rules.

2. Nature and Availability of Expertise

This scale is composed of six subscales, each concerned with an aspect of expert knowledge:

- Nature of Expertise

This subscale contrasts verbal and performance-based expertise. Verbal expertise is relatively easy to acquire using well understood knowledge acquisition techniques employing interviews and the analysis of documents. Performance-based expertise is more difficult to acquire, usually requiring the use of complex simulations.

Low Risk:	Easy to express verbally.
Low-Moderate Risk:	Expressible verbally, but difficult.
Moderate-High Risk:	Partially expressible verbally, requires some performance.
High Risk:	Performance-based expression, verbalization extremely difficult and unreliable.

- Availability

Availability of expertise depends on the presence or absence of operational experience. Advanced Air Force systems may not have operational expertise gained through experience solving a variety of real problems. In the absence of operational expertise, analytic or theoretical expertise may be available, which increases risk because the expertise is "untested." If no expertise exists, either operational or analytic, implementation risk is extremely high; absence of expertise is usually a "showstopper" because it indicates that a solution for the problem is unknown.

Low Risk:	Significant operational expertise exists.
Low-Moderate Risk:	Moderate operational expertise exists.
Moderate-High Risk:	No operational expertise; only analytic.
High Risk:	No operational or analytic expertise exists.

- Cases

Representative cases or operational scenarios are needed to design, develop, test, and evaluate a system. If cases do not exist, they may be created or compiled with varying cost in time and resources. If it is impossible to generate operational cases, risk is extremely high; as with the absence of expertise, the absence of cases suggests that a solution for the problem is unknown.

Low Risk:	Cases already exist and provide good coverage of all situations.
Low-Moderate Risk:	None exist; readily created/compiled.
Moderate Risk:	None exist; significant resources needed to create/compile.

Moderate-High Risk: None exist; hypothetical cases could be created.

High Risk: None exist; cases cannot be created.

- Expert Agreement

The development of an intelligent module is facilitated when different experts approach problems in the same way and when the results produced by different experts are similar. In addition, the acceptability of an application is usually low when experts do not agree. These risks increase as agreement among experts decreases.

Low Risk: High agreement.

Moderate Risk: Moderate agreement.

High Risk: Little or no agreement.

- Domain Coverage

Intelligent systems are easier to design and develop when there is a single expert who can cover the entire application domain. If domain expertise is distributed across several experts, knowledge acquisition risk increases. This risk can be reduced somewhat if a single expert is able to represent/arbitrate multiple experts.

Low Risk: All experts cover entire domain.

Moderate Risk: Experts specialize in areas of domain, but some experts know sources in all areas and can represent/arbitrate these sources.

High Risk: Experts specialize; none can represent the entire domain.

- Volatility of Knowledge

When knowledge in a domain is unstable, the likelihood of capturing inappropriate, out-of-date knowledge increases, as does the difficulty of maintaining the knowledge base. These factors increase the risk of producing an unacceptable system, both in terms of functionality and cost.

Low Risk: Expertise is stable; knowledge acquired in prior years will be essentially the same as today's knowledge.

Moderate Risk: Expertise is moderately volatile.

High Risk: Expertise is evolving quickly.

3. Complexity/Difficulty of Task

For expert system applications, problems that are cognitively difficult for a human expert tend to be difficult for a intelligent system. This scale measures the difficulty of

the application task for a human expert.

Low Risk:	Straightforward for any practitioner.
Low-Moderate Risk:	Straightforward for an expert.
Moderate-High Risk:	Moderately difficult for an expert.
High Risk:	Very difficult for expert; an expert is often unsuccessful.

4. Role of System

Risk increases as the role of the intelligent system approaches that of an autonomous decision maker because the knowledge used by such systems must be complete and verified, qualities that increase the difficulty and risk of system design and development. Systems that advise competent human decision makers can serve a useful function with less complete and verified knowledge.

Low Risk:	Consultation/Advisory system; aid to competent human decision maker.
Moderate Risk:	System is the decision maker, with a lower-skilled human in the loop.
High Risk:	Closed-loop autonomous system.

5. Size and Complexity of Knowledge Base

This scale is composed of four subscales, each concerned with an aspect of size and complexity of the knowledge base:

- Size

Large knowledge bases increase risks associated with hardware, verification and validation, maintenance, and performance. The larger the knowledge base, the greater the risk. The metric for knowledge base size is number of knowledge base elements (rules, objects, facts, attributes, and key relationships).

Low Risk:	Very Small (< 50 elements).
Low-Moderate Risk	Small (~ 200 elements).
Moderate Risk:	Medium (~ 400 to ~ 700 elements).
Moderate-High Risk:	Large (~ 1000 elements).
High Risk:	Very Large (10,000 + elements).

- Interrelatedness of Knowledge Base Elements

"Interrelatedness" is a measure of the complexity of the knowledge base. The greater the interrelatedness of knowledge base elements, the greater the "ripple"

effect of changes in elements, and the greater the difficulty of knowledge engineering and knowledge base maintenance.

Low Risk:	Most elements independent; changes in one element do not cause changes in others.
Moderate Risk:	Some coupling and interdependence among elements.
High Risk:	Changes in one element cause ripples of change through other elements.

- Knowledge Base Prototypability

Another aspect of the size and complexity of the knowledge base is its prototypability. Knowledge-based systems should be developed incrementally, beginning with a relatively small and simple knowledge base and increasing its depth and breadth with each prototype iteration. A prototype, though small and simple, should be meaningful, however, in the sense of providing useful information and a convincing demonstration of concepts. This scale measures the ability to define a meaningful prototype that is less than the full system in knowledge base size and complexity.

Low Risk:	Meaningful prototype can be defined that is less than the full system in depth and breadth.
Low-Moderate Risk:	Need full depth for a prototype (complete functionality).
Moderate-High Risk:	Need full breadth for a prototype (complete scope).
High Risk:	No meaningful prototype can be defined that is less than full scope and functionality.

- Knowledge Base Scalability

"Scalability" refers to the ability to extend the prototype intelligent module to the full module. Scaling to the full scope and functionality is relatively easy when no redesign is required and knowledge base expansion is moderate. Risk increases as the magnitude of knowledge base augmentation increases and if a redesign is required.

Low Risk:	No redesign is required and only moderate knowledge base augmentation is needed to increase scope and functionality.
Moderate Risk:	No redesign is required, but substantial knowledge base augmentation is needed to increase scope and functionality.
High Risk:	Redesign is required to increase scope and functionality.

6. Applicability of Current AI Tools

AI tools greatly speed the development of intelligent applications by providing "canned" reasoning frameworks, knowledge base facilities, and explanation capabilities. Supported commercial tools, if appropriate for the application, reduce risk substantially. Risk increases when non-supported "research" tools are used and when extensions to tools are required. Risk is, of course, high when no appropriate tools are available.

Low Risk:	Current commercial tools are appropriate, without extensions.
Low-Moderate Risk:	Current research tools are appropriate, without extensions.
Moderate-High Risk:	Require extensions to current tools.
High Risk:	No current tools are appropriate, even with extensions.

7. Advanced Technology Requirements

Applications may depend on advanced, research technology. Risk increases as dependence on advanced technology increases.

Low Risk:	None; the application can be developed using well understood technology.
Moderate Risk:	Optional; the application does not require advanced technology, but it provides opportunities for the use of advanced technologies and applied R&D.
High Risk:	Required; the application cannot be built without advanced technology.

Systems Engineering Risk

Many complex applications require more than a single intelligent module. These applications require the integration of intelligent modules into a system composed of conventional software components such as databases, analysis routines based on operations research techniques, statistical packages, and the like. Systems Engineering Risk is the application risk associated with combining intelligent modules with conventional modules to build a large and complex intelligent system. Subscales measure intelligent system qualities that experience has shown to be indicative of systems engineering risk.

1. AI/Conventional Module Mix

This subscale measures the risk produced by the mix of intelligent and conventional modules. Risk increases with the number of intelligent modules and with the introduction of conventional modules to the intelligent system.

Low Risk:	Single intelligent module.

Low-Moderate Risk:	Single intelligent module plus conventional modules.
Moderate-High Risk:	Multiple intelligent modules.
High Risk:	Multiple intelligent modules plus conventional modules.

2. Degree of Integration

"Integration" is measured by the frequency and amount of interaction among system components. The greater the interaction, the greater the system complexity, and the greater the risk.

Low Risk:	Low interaction among modules.
Moderate Risk:	Moderate interaction among modules.
High Risk:	High interaction among modules.

3. System Scalability

One of the scales measuring AI Risk dealt with the ease of extending a prototype AI module to a full module. The present scale is concerned with the ability to extend the prototype intelligent system composed of several modules to the full system. Scaling to the full system is relatively easy when no redesign is required and augmentation of the system is moderate. Risk increases as the magnitude of the augmentation increases and if a redesign of the system is required.

Low Risk:	No redesign is required and only moderate augmentation is needed to increase scope and functionality.
Moderate Risk:	No redesign is required, but substantial augmentation is needed to increase scope and functionality.
High Risk:	Redesign is required to increase scope and functionality.

4. Response Performance Requirements

Some applications have no specified response performance requirements or require response times measured in minutes. Others demand response times in milliseconds. Current capabilities can provide response times in seconds and certainly in minutes. Millisecond performance may require special hardware and compilation techniques.

Low Risk:	No response time requirement.
Low-Moderate Risk:	Response performance in minutes.
Moderate-High Risk:	Response performance in seconds.
High Risk:	Response performance in milliseconds.

5. Hardware Requirements: Development and Delivery

The ideal hardware requirement is, of course, no requirement. If no hardware is specified, hardware producing the lowest risk can be chosen. When hardware is specified, it is usually the case that AI hardware offers the most efficient development environment and lowest risk delivery environment for AI modules, with risk increasing when off-the-shelf non-AI hardware is specified. For the conventional elements of intelligent systems, this relationship is reversed: risk is usually lower with non-AI hardware. The highest risk condition for AI modules and conventional modules is the specification of special hardware; in this case, software development risk is compounded by the need to run on new hardware. Requirements for development hardware and delivery hardware are rated separately using the scale points below.

Low Risk:	None specified.
Low-Moderate Risk:	Specified AI hardware for AI modules / non-AI hardware for conventional modules.
Moderate Risk:	Non-specified non-AI hardware for AI modules / non specified AI hardware for conventional modules.
Moderate-High Risk:	Specified non-AI hardware for AI modules / specified AI hardware for conventional modules.
High Risk:	Special hardware.

6. Software Requirements: Development and Delivery

As with hardware, the ideal software requirement is no requirement. When software is specified, AI languages or tools, e.g., LISP, PROLOG, KEE, usually reduce the risk for AI modules. Non-AI languages are generally believed to reduce the risk for conventional modules, although this is arguable. The highest risk condition for AI modules and conventional modules is generally thought to be the specification of a language or development environment for which there is relatively little practical experience in the development of intelligent systems, e.g., ADA. Requirements for development software and delivery software are rated separately using the scale points below.

Low Risk:	None specified.
Low-Moderate Risk:	Specified AI software for AI modules / non-AI software for conventional modules.
Moderate-High Risk:	Specified non-AI software for AI modules / specified AI software for conventional modules.
High Risk:	Specified "new" software development environment, e.g., ADA.

7. Maintainability

System maintenance is a risk category not usually considered in evaluating systems. It is important because of its potentially significant impact on the usability and life-cycle cost of systems. Maintainability is measured in terms of the stability of the data, models, and components of the intelligent system. The greater the frequency and magnitude of changes, the lower the usability of the system, the greater the cost of maintaining the system, and the greater the risk of failure.

Low Risk:	None; once built, the system will require little maintenance.
Moderate Risk:	Moderate; system maintenance will approach but not exceed system development effort.
High Risk:	HIgh; system maintenance will exceed system development effort.

Value & Cost

The value provided by an application and the cost of the application are obviously key factors in application screening. Value and cost are collapsed into a single dimension, with value less than or equal to life cycle cost at the left or low end of the dimension and value "far exceeding" life cycle cost at the right or high end. "Far exceeding" is not defined; the analysis of value and cost for an application screening is intended to be crude, providing only a very rough order of magnitude estimate of the anticipated "bottom line." The analysis is conducted through the use of four subscales, three examining indexes of value and one attempting to represent the cost of developing a prototype system and a full system.

1. Economic Value

   This subscale measures the estimated value of the application in terms of economic factors such as cost savings and increased revenue.

Low Value:	Payoff less than or equal to life cycle costs of the system.
Moderate Value:	Payoff "exceeding" life cycle costs.
High Value:	Payoff "far exceeding" life cycle costs.

2. Effectiveness

   This subscale measures the value of the application in less tangible terms than cost savings. "Effectiveness" is concerned with the "quality" factors such as mission effectiveness, survivability, and reliability.

Low Value:	Low effectiveness.
Moderate Value:	Moderate effectiveness.
High Value:	High effectiveness.

3. Generality

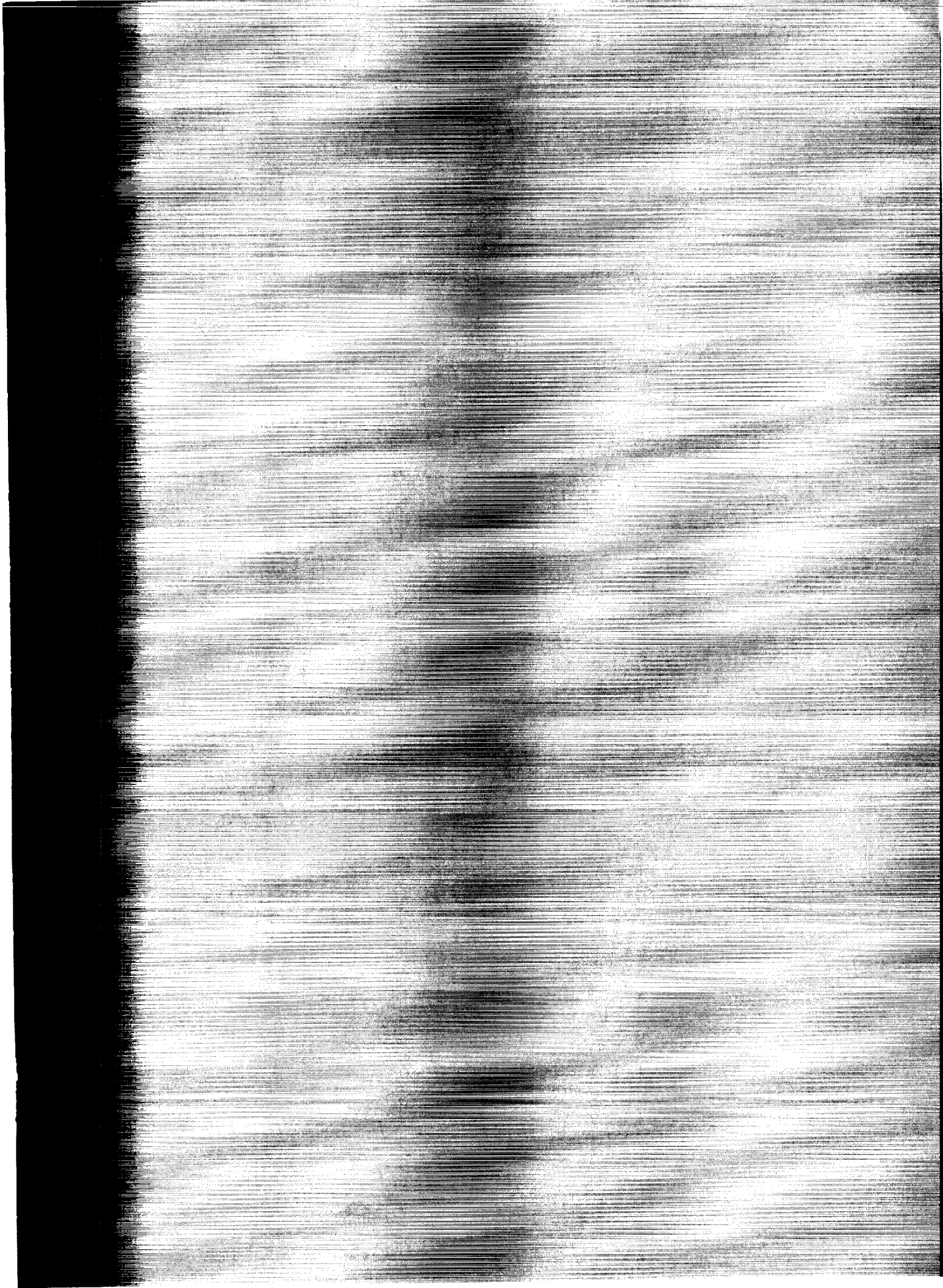
The value of an application can go beyond its specific domain. The concepts and techniques created in designing and developing the application could be used to leverage the design and development of different applications. The basic structure, implementation framework, or knowledge used in the application could be applied to similar problems in the organization sponsoring the application or in other parts of NASA. The greater the generality of an application, the greater its value.

Low Value:	Low generality; one-time use for the system.
Moderate Value:	Moderate generality; some concepts or techniques could be generalized for use in some applications.
High Value:	High generality; details of the system could be applied to many applications.

3. Cost

The cost of an intelligent system is measured in estimated person-years of effort required to produce conventional components, AI components, and any applied R&D required by the system. Cost estimates for a prototype system and for a full system are rated separately for each of the three work elements (conventional components, AI components, and applied R&D) using the scale points below.

Low Cost:	0 person-years.
Low-Moderate Cost:	1 person-year.
Moderate Cost:	5 person-years.
Moderate-High Cost:	10 person-years.
High Cost:	20 person-years.



Requirements Specification

Knowledge-Based Decision-Support System for SSF Engineering Managers

Submitted By

ISX Corporation
501 Marin St., Suite 214
Thousand Oaks, CA 91360

May 11, 1990

Funded By

NASA Ames Research Center
Small Business Innovative Research Contract NAS2-13161

Table of Contents

1.0 Introduction

1.1 Purpose and Scope

This document describes requirements for a knowledge-based decision-support system for Space Station Freedom (SSF) engineering managers. The target application is SSF assembly sequence planning. Development of these requirements is funded by NASA Ames Research Center (ARC) Phase I SBIR (Small Business Innovative Research) Contract NAS2-13161.

1.2 Identification

This is the first released version of the requirements specification for a knowledge-based decision-support system for SSF engineering managers. It is designated version 1.0 and is dated May 11, 1990.

This work was performed using ISX's Intelligent Systems Engineering (ISE) methodology. A key feature of the ISE methodology is the early and continuous use of prototypes to inform the development of requirements and the system design. The requirements described in this document, and the preliminary design described in the design document cited in Section 2, were developed with the support of prototypes used to express and test assumptions and approaches. As additional prototypes are developed, requirements and the design will also continue to develop, increasing in both depth and breadth. These enhancements and extensions will be documented in subsequent drafts of this requirements specification and the design document, which will be issued periodically as the products of work performed under a subsequent contract.

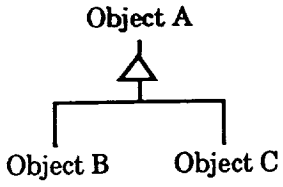
1.3 Acronyms

AC	Assembly Complete
ARC	NASA's Ames Research Center
ASRM	Advanced Solid Rocket Motor
CG	Center of Gravity
FEL	First Element Launch
ISE	Intelligent Systems Engineering
IVA	Intravehicular Activity
EVA	Extravehicular activity
MAX	Manager's Assistant
MTC	Man-Tended Capability
NSTS	National Space Transportation System (the Shuttle)
PAA	Personal Analysis Assistant

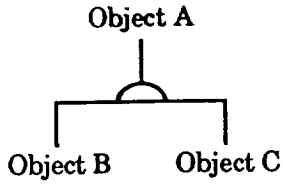
©ISX Corporation

PMC	Permanent Manned Capability
RMS	Remote Manipulator System
SBIR	Small Business Innovative Research
SSF	Space Station Freedom
SSFPO	Space Station Freedom Program Office

1.4 Notation



Assembly Structure: Object B *is part of* Object A



Hierarchical Classification Structure: Object B *is an* Object A

Instance Connection: 1 Object A is connected to many Object Bs

©ISX Corporation

1.5 Background

This work is funded by ARC Phase I SBIR Contract NAS2-13161. The objective is the development of specifications for a knowledge-based decision-support system for Space Station Freedom (SSF) engineering managers.

Following meetings with Paul Neumann and Ben Barker of the Space Station Freedom Program Office (SSFPO) and conversations with Henry Lum of ARC and Gregg Swietek of SSFPO, it was determined that SSF assembly sequence planning is an appropriate application for the Phase I SBIR, and that the SSFPO is sufficiently interested to provide staff to support application assessment, requirements analysis, and design activities. The customer and expert for the proposed application is Bill Bastedo of the SSFPO. Mr. Bastedo offered two days of his and his staff's time to support an application assessment, followed by occasional review of requirements and design documentation.

Based on experience derived from hundreds of projects conducted over the past seven years, ISX has developed an Intelligent Systems Engineering (ISE) methodology that combines the methods of systems engineering and knowledge engineering to meet the special systems development requirements posed by *intelligent systems*, systems that blend artificial intelligence and other advanced technologies with more conventional computing technologies. The ISE methodology defines a phased program process that begins with application screenings designed to provide a preliminary determination of the relative technical risks and payoffs associated with a potential application, and then moves through application assessments to requirements definition, design, and development.

Using information provided by Paul Neumann and Ben Barker, ISX completed the application screening in December, 1989. The next step, performed on February 26 and 27, 1990, was the application assessment. The assessment team included Bill Bewley, Gary Edwards, David Rosenberg, and Allen Smith of ISX. Peter Warren and Brook Sullivan of Booz-Allen & Hamilton were the application experts, and Bill Bastedo of the SSFPO provided application information in the form of feedback to a presentation of preliminary assessment results on the afternoon of February 27.

The document "Application Assessment Report: Space Station Assembly Sequence Planning," cited in Section 2, presents assessment results in detail. These results can be briefly summarized as follows:

- **Analysis of the Process.** SSF assembly sequence planning is an integral and vital component of the ongoing SSF program. Its function, broadly stated, is to define launch vehicle flights and manifests that satisfy a complex, interdependent set of SSF, launch vehicle, and programmatic constraints. In both development of a new baseline plan and the performance of what-ifs, the planner is charged with developing the manifest, defining cargo elements, placing cargo elements in the NSTS cargo bay, calculating performance measures, validating the measures, identifying violated constraints, and revising invalidated parts of the plan. The analyses supporting these activities are time-consuming and must be performed under severe time pressure. The what-ifs are especially difficult in that time pressure is always great, they occur frequently, and the timing of their occurrence is difficult to predict. Because of the time pressure, planners find it difficult to perform the depth and breadth of analysis required to produce accurate and easily justifiable results.

- **Preliminary System Concept.** The load on the assembly sequence planner can be reduced by providing a personal "assistant" in a machine that would perform the work

©ISX Corporation

that is currently so tedious and time-consuming for the human planner. The assistant would be a "Personal Analysis Assistant" that helps the human planner by doing the bookkeeping to maintain plan data and executing the procedures and heuristics currently used by the human planner to define flights, develop flight manifests, define and place cargo elements, calculate performance measures, and identify violated constraints. The Personal Analysis Assistant (PAA) system would consist of three major elements: (1) an Infrastructure for analysis support and validity checking; (2) an Intra-Flight Value-Added function that would generate and place cargo elements given a manifest; and (3) an Inter-Flight Value-Added function that would generate a manifest given major milestones.

- **Assessment of Value and Feasibility.** The assessment indicated that a Personal Analysis Assistant system for assembly sequence planners was both feasible and valuable. The system concept was rated on four dimensions: AI Risk, Systems Engineering Risk, Value, and Cost. *AI Risk* was judged low to moderate because of the ready availability of expertise, the anticipated moderate size of the knowledge base, the applicability of current AI tools, and the assessment/planning nature of the application, for which a variety of proven AI techniques are available. *Systems Engineering Risk* was judged moderate to high because of strong requirements for scaleability and integration of intelligent and conventional components, a risk which is partially mitigated by a system concept that supports extensibility by SSFPO and Booz-Allen & Hamilton staff. *System Value* was rated high because the system is expected to greatly enhance assembly sequence planning, an extremely high-value task, by supporting more timely, less costly, and more effective planning and what-if analyses. *System Cost* was estimated to be low to moderate, with development of AI components expected to require approximately 1 person-year and systems engineering cost estimated to be approximately 5 person-years

The requirements analysis for the Personal Analysis Assistant followed the application assessment. The ISE requirements analysis process begins with an overview of the application and a definition of the "system vision," which is a top-level view of the proposed intelligent system and its role in the application. This is followed by identification of objects involved in the application, development of operational scenarios, and identification of detailed object descriptions, including structural relationships, attributes, and processing. The system overview is described in Section 3. Section 4 describes top-level objects identified for the application. Section 5 summarizes the operational scenarios. Detailed object descriptions are presented in Section 6.

2.0 References

Application Assessment Report: Space Station Assembly Sequence Planning, ISX Corporation. March 6, 1990.

Kaidy, James T., and Bastedo, William G. Space Station Assembly Sequence Planning: An engineering and operational challenge. *Proceedings of the AIAA*, 1988.

Space Station Stage Summary Databook. Space Station Freedom Program Office, December 15, 1989.

Warren, P. *Baseline Assembly Sequence Rationale*, February 28, 1990.

Warren, P. and Sullivan, B. *ASRM Trade Study*, 1990.

©ISX Corporation

3.0 System Overview

This section presents an overview of the assembly sequence planning application and a definition of the preliminary system vision.

3.1 Overview of the Application

Figure 1 presents a graphic overview of the SSF assembly sequence planning process. The process has two phases. The first phase is performed by SSFPO and Booz-Allen & Hamilton staff and involves development of the flight manifest (the inter-flight plan), definition and placement of cargo elements in the NSTS cargo bay (intra-flight plans), and documentation and publication of plans. This phase iterates in frequent "Internal Iteration" loops from plan validation by internal (SSFPO) analyses back to plan revision. The second phase is performed by Space Station Freedom (SSF) engineering groups, which evaluate the documented plans, identify constraint violations, and provide feedback to SSFPO. This phase is characterized by less frequent "External Iterations" from plan validation by external (SSF engineering) analyses to plan revision by SSFPO.

The process begins with definition of constraints in three categories:

- National Space Transportation System (NSTS) constraints, e.g., volume and mass capacity, flight rate.
- SSF hardware constraints, e.g., configuration and assembly elements.
- Programmatic constraints, e.g., major program milestones and priorities.

Given a space station configuration description (complete with definitions for all assembly elements) and major milestone constraints, e.g., dates and capability requirements for First Element Launch (FEL), Man-Tended Capability (MTC), Permanent Manned Capability (PMC), and Assembly Complete (AC), the SSFPO assembly sequence planner develops the flight manifest, which consists of determining the number and timing of flights and assigning assembly elements to flights. Following definition of the manifest, intra-flight planning defines cargo elements as compositions of assembly elements manifested to the flight and places cargo elements in the cargo bay.

The manifest and its associated intra-flight cargo-element groupings and cargo bay placements are in essence a "hypothesis" which is tested by comparing performance measures derived from the hypothesis with standards defined by programmatic/milestone constraints, NSTS constraints, and SSF hardware constraints. Measures include mass, volume, center of gravity (CG), the power requirement, the intra- and extra-vehicular activity (IVA and EVA) requirements, and the Remote Manipulator System (RMS) reach requirement. If measures and/or margins violate constraints, the invalidated part of the plan is revised by changing/relaxing constraints or changing elements of the plan: the manifest, cargo element definition, or cargo bay placement.

©ISX Corporation

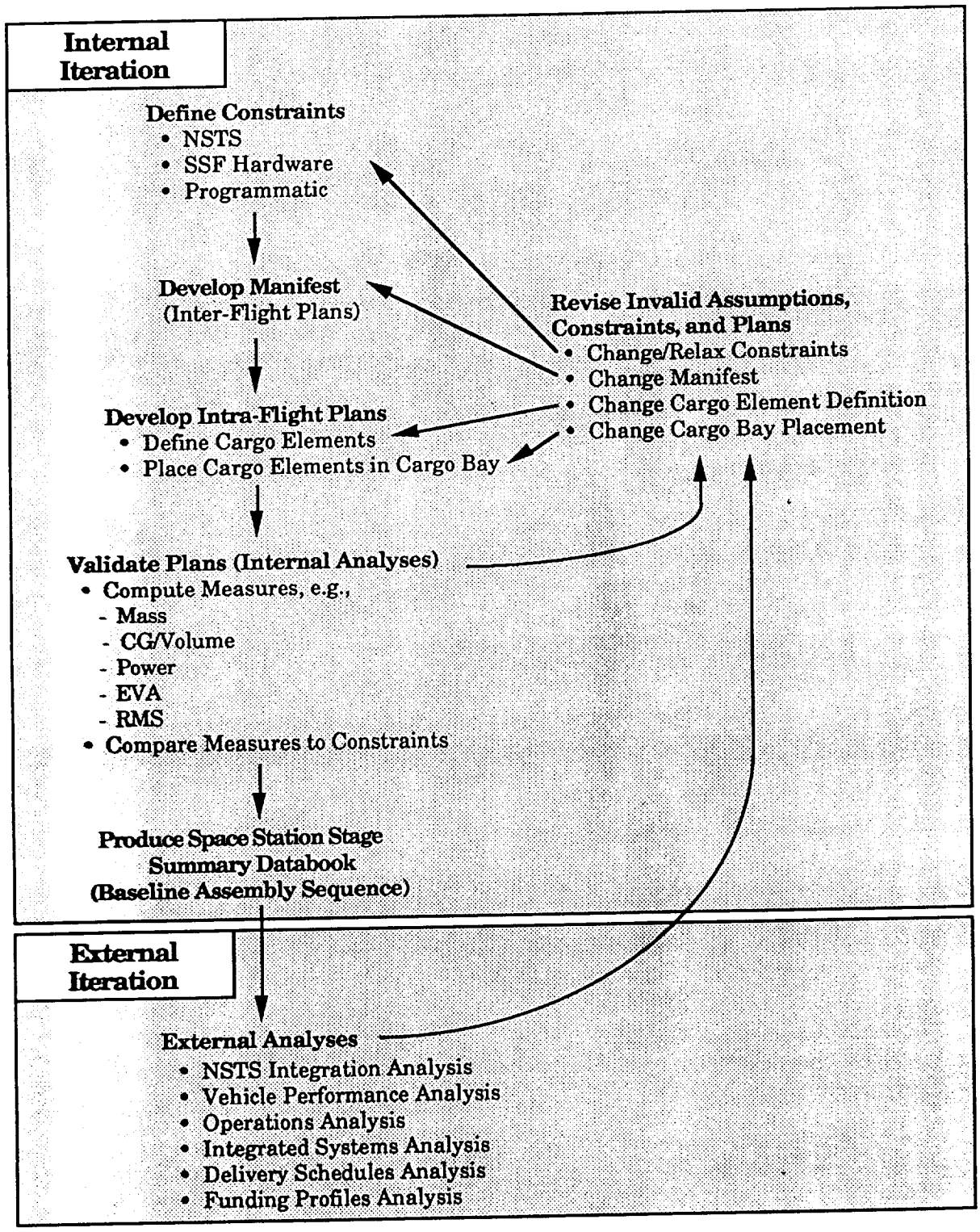


Figure 1: The SSF Assembly Sequence Planning Process

©ISX Corporation

When the SSFPO is satisfied with the validity of plans (normally after informal consultation with external groups), they are documented in a Space Station Stage Summary Databook, which is distributed for review to SSF engineering groups, who perform analyses and identify constraint violations. SSFPO collects feedback on constraint violations and uses the feedback to drive plan revisions.

Although this description may suggest that assembly sequence planning begins anew with each planning cycle, with constraints and plans being defined and developed from scratch on each iteration, assembly sequence planning is in fact an ongoing process in which constraints and plans are revisions of prior constraint definitions and plans.

There are two major areas in which the assembly sequence planning process is problematic: the assembly sequence planner's work load and the maintenance of plan data. Much of the assembly sequence planner's load is attributable to tedious and time-consuming analyses that involve executing procedures, applying well-known heuristics, and processing the horrendous detail of plan data, including assembly elements, constraints, and all their interdependencies. This is the intellectual "scut work" of analysis and planning, and the load it imposes often prevents the planner from spending time doing the creative problem solving required for assembly sequence planning. It also makes it difficult for the planner to provide quick responses to questions and to perform the depth and breadth of analysis needed to produce a new baseline plan or evaluate a what-if.

3.2 System Vision

There is an opportunity to unload the planner by providing a personal "assistant" in a machine that would perform the work that is currently so tedious and time-consuming. The assistant would be a "Personal Analysis Assistant" that helps the human planner by doing the bookkeeping to maintain plan data and executing the procedures and heuristics currently used by the human planner to define flights, develop flight manifests, define and place cargo elements, calculate performance measures, and identify violated constraints. This unloading would speed the planning process, enable greater depth and breadth of analysis, and free the human planner to spend more time doing what only the human planner can do: evaluating analysis results; revising invalid assumptions, constraints and plans; generating new solutions to assembly sequence planning; and testing solution hypotheses with what-ifs.

Assembly sequence planning is a difficult problem for humans, and full automation is beyond the capability of current technology (and is probably not desirable, in any case). The system is, therefore, intended to assist the human planner, not to automate the assembly sequence planning process. The planner will specify the tasks the system is to perform, e.g., validate specified flight plans, define cargo elements and place them in the cargo bay of a flight, or define flights and flight manifests for specified major milestones. The planner will also specify the assumptions the system is to use, e.g., flight plans, assembly elements, manifests, and constraints. Given these specifications, the system will generate results. The planner will then review the results, possibly request explanations of results or follow-up analyses, and then make the final decisions, potentially modifying the computer-generated plan. The system is essentially a laboratory for the assembly sequence planner, providing an environment in which the planner can define and run conceptual experiments that help him do the problem solving needed to generate plans.

The Personal Analysis Assistant (PAA) is one of several "assistant" systems built on the Manager's Assistant (MAX) framework. The MAX framework, which is an infrastructure

©ISX Corporation

supporting the development of specific assistant systems, and the PAA will be developed concurrently but under separate funding. The infrastructure will support both system developers in producing the specific assistant system and user organizations in maintaining and extending system knowledge bases.

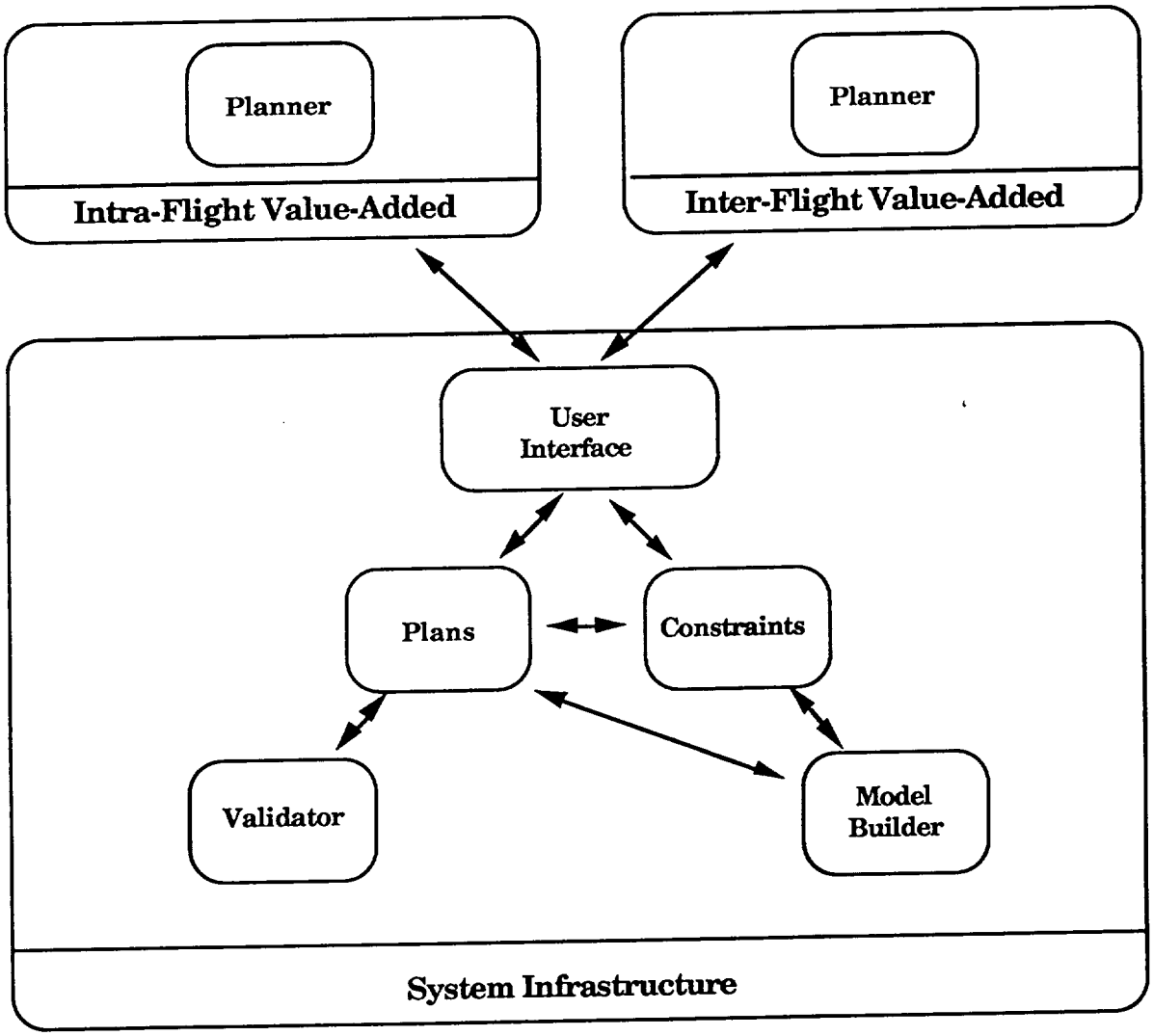


Figure 2: Top-Level Architecture of the Personal Analysis Assistant

Figure 2 shows the top-level architecture of the Personal Analysis Assistant (PAA) system. The functions shown in the figure are grouped into three major software elements: the System Infrastructure, Intra-Flight Value-Added, and Inter-Flight Value-Added.

- **The System Infrastructure** provides the capability for building representations of assembly sequence plans, e.g., flights, manifests, cargo elements, and associated performance measures; networks of dependencies among plan objects; constraint networks describing bounds on measures; and user-specifications of constraints and analysis options. It also supports calculation of performance measures and

©ISX Corporation

identification of violated constraints, and intra-flight plans. This corresponds to "Validate Plans" in Figure 1. Finally, it is the substrate on which functions providing inter-flight and intra-flight value-added functionality are built.

- **Intra-Flight Value-Added** provides functionality that will, given the manifest for each flight, define cargo elements, place them in the NSTS cargo bay, compute performance measure values, and identify violated constraints. This corresponds to support of "Develop Intra-Flight Plans" and "Validate Plans" in Figure 1.

- **Inter-Flight Value-Added** provides functionality that will, given major milestone dates and capability requirements, determine the number and dates of required flights and develop a manifest for each flight.

As shown in Figure 3, the system is evolutionary in that it is based on the System Infrastructure, which provides value to the planner at the earliest stages of development and builds on that value by supporting the addition of new functionality and the enhancement of existing functionality over the life of the system.

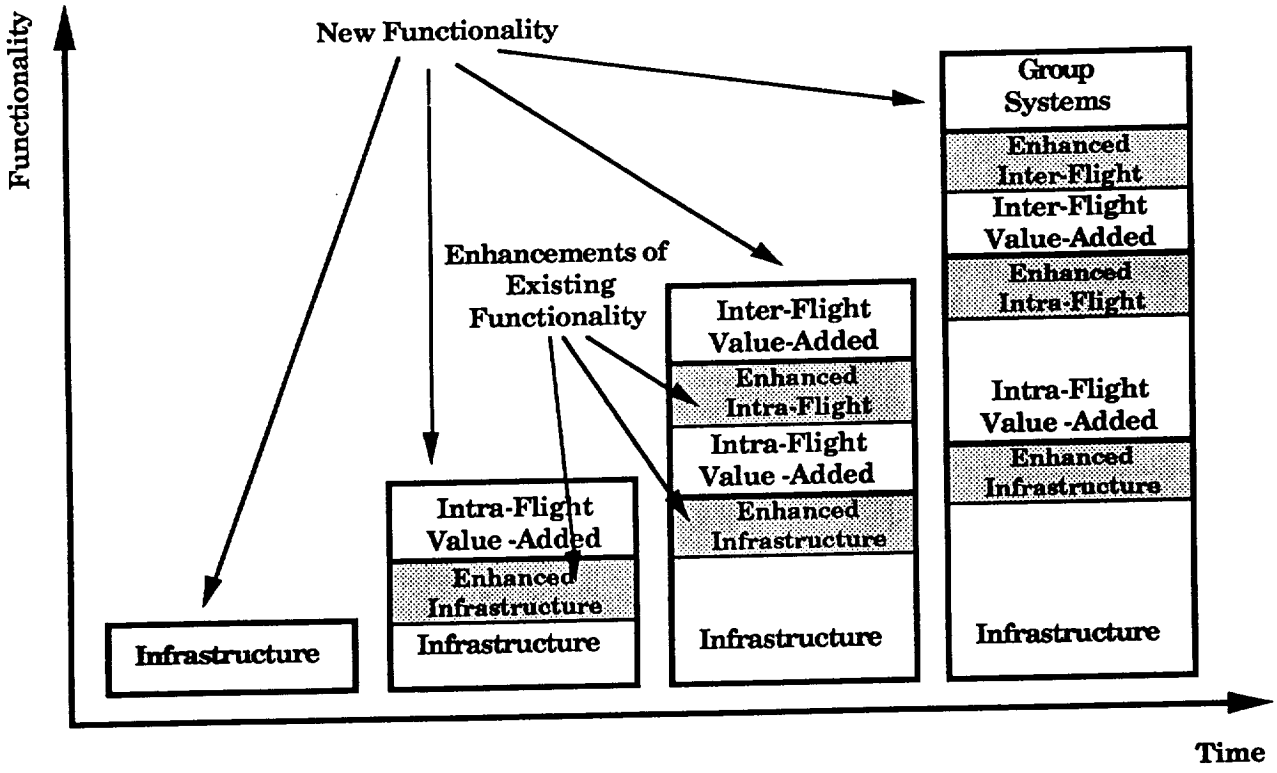


Figure 3: The System Concept is Evolutionary, Providing Early Value and Supporting the Addition and Enhancement of Functionality.

©ISX Corporation

4.0 Top-Level Objects

Application objects are grouped into seven top-level objects: Assembly Sequence Planners (system users), the Personal Analysis Assistant, Information Sources, Constraints, Plan Elements, Work Products, and Information Consumers. These objects and their top-level relationships are shown in Figure 4. The connecting arrows denote message connections between the objects, connections in which a "sender" sends a message to a "receiver" in order to cause processing to be done by the receiver.

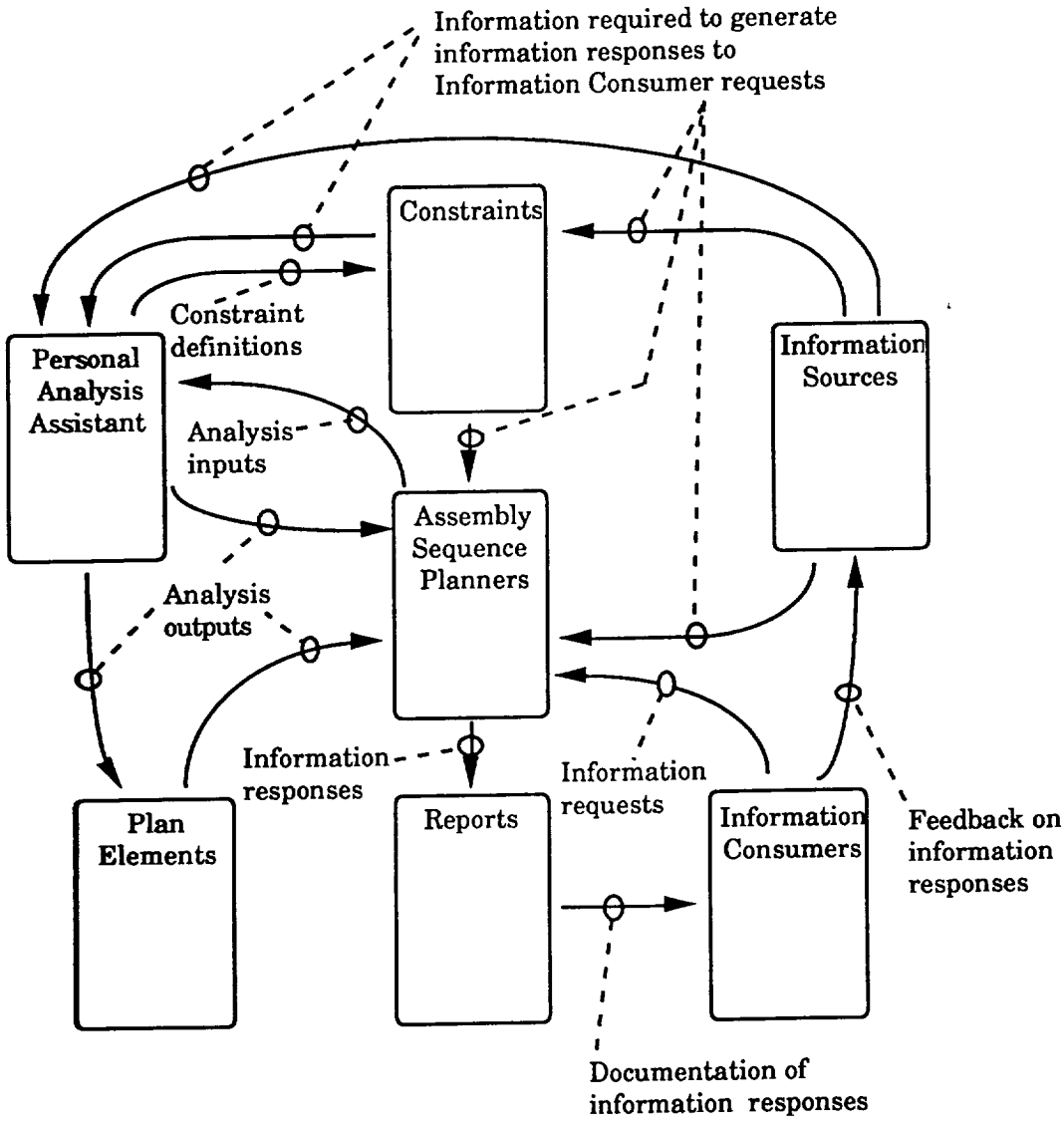


Figure 4: Top-Level Objects and Message Connections

Top-level objects are defined in the following paragraphs. Detailed descriptions of lower-level objects, including structural relationships, attributes and the processing to be performed by objects, are presented in Section 6.

©ISX Corporation

Object: Assembly Sequence Planners (PAA system users)
Description: SSFPO staff and contractors who, given requests for information from Information Consumers and information supplied by Information Sources, generate information required by Information Consumers.

Object: Personal Analysis Asssistant
Description: A software system that supports Assembly Sequence Planners in developing new baseline assembly sequence plans and in performing what-if analyses that provide responses to questions on proposed perturbations of the current baseline.

Object: Information Sources
Description: Sources of information, including human sources, databases, and computational tools, that provide information needed to generate information requested by Information Consumers.

Object: Constraints
Description: Constraints on the development of an assembly sequence plan, including programmatic constraints and constraints associated with the NSTS and the SSF hardware configuration.

Object: Plan Elements
Description: Parts of assembly sequence plans, including launch manifests, flights, cargo elements, and the results of analyses measuring plan goodness.

Object: Reports
Description: Documents that provide information to SSF Information Consumers on the baseline assembly sequence and preliminary stage definition or in response to questions posed by Information Consumers regarding proposed perturbations of the baseline assembly sequence.

Object: Information Consumers
Description: Persons who use information generated by SSFPO Information Analysts, as documented in the Space Station Stage Summary Databook or What-If Reports. They are representatives of SSF functional organizations.

5.0 Operational Scenarios

There are two classes of operational scenarios relevant to assembly sequence planning:

- Development of a new baseline assembly sequence plan for documentation in the Space Station Stage Summary Databook.
- Performing what-if analyses that provide responses to questions on proposed perturbations of the current baseline, e.g., "What if the NSTS mass capacity was increased by 13,000 lbs?".

This section describes PAA operational scenarios in each of these classes. For each scenario, the scenario class, enabling software elements, the goal or or capability illustrated by the scenario, and assumed prior conditions are identified, followed by a specification of process steps.

©ISX Corporation

5.1 Baseline Development Scenarios

Scenario 1

Class: Baseline Development
Enabler: System Infrastructure
Goal: User definition of constraints, inter-flight plans, and intra-flight plans; PAA validation of plans
Conditions: The user knows the following:

- the SSF configuration, including definition of assembly elements
- NSTS constraints, including standards for mass, volume, center of gravity (CG), Remote Manipulator System (RMS) utilization, and EVA
- major milestones, i.e., dates for milestones such as Man-Tended Capability (MTC), Permanent Manned Capability (PMC), and Assembly Complete (AC)
- dates and definition of phases within major milestones, e.g., Spacecraft Activation, Primary Hardware Control, and International Element Delivery
- priorities for each phase, e.g., spacecraft activation, health and status data to the ground, respond to command and control signals, and insure continuance of assembly -- all priorities for the Spacecraft Activation phase
- the flight schedule
- cargo element definition
- cargo element assignment to flights
- cargo element placement in the NSTS cargo bay

Process: The PAA/user system performs the following:

- the user enters assembly elements, NSTS constraints, milestones, phases, priorities into the PAA (constraints)
- the user enters number and timing of flights, assignment of assembly elements to flights into the PAA (inter-flight plans)
- the user enters cargo elements and placement of cargo elements in the NSTS cargo bay into the PAA (intra-flight plans)
- the PAA computes values for measures including mass, volume, center of gravity (CG), Remote Manipulator System (RMS) utilization, EVA requirements, IVA requirements, operability, reliability.
- the PAA compares computed values to standards defined by constraints and identifies and assesses criticality of violated constraints
- the PAA reports violated constraints and estimated criticality to the user
- the user modifies any or all of the givens defined in Conditions and reruns the process

Scenario 2

Class: Baseline Development
Enabler: System Infrastructure, Intra-Flight Value Added
Goal: User definition of constraints and inter-flight plans; PAA definition of intra-flight plans and validation of plans
Conditions: The user knows the following:

©ISX Corporation

- the SSF configuration, including definition of assembly elements
- NSTS constraints, including standards for mass, volume, center of gravity (CG), Remote Manipulator System (RMS) utilization, and EVA
- major milestones, i.e., dates for milestones such as Man-Tended Capability (MTC), Permanent Manned Capability (PMC), and Assembly Complete (AC)
- dates and definition of phases within major milestones, e.g., Spacecraft Activation, Primary Hardware Control, and International Element Delivery
- priorities for each phase, e.g., spacecraft activation, health and status data to the ground, respond to command and control signals, and insure continuance of assembly -- all priorities for the Spacecraft Activation phase
- the flight schedule

Process: The PAA/user system performs the following:

- the user enters assembly elements, NSTS constraints, milestones, phases, priorities into the PAA (constraints)
- the user enters number and timing of flights, assignment of assembly elements to flights into the PAA (inter-flight plans)
- the PAA defines cargo elements
- the PAA places cargo elements in the NSTS cargo bay for each flight
- the PAA computes values for measures including mass, volume, center of gravity (CG), Remote Manipulator System (RMS) utilization, EVA requirements, IVA requirements, operability, reliability.
- the PAA compares computed values to standards defined by constraints and identifies and assesses criticality of violated constraints
- the PAA identifies and prioritizes potential intra-flight solutions to constraint-violation problems
- the PAA develops task plans for implementing the intra-flight solutions
- the PAA reports violated constraints, estimated criticality of violations, and suggested intra-flight solutions to the user
- the user tells the PAA to implement the high priority suggested solution OR the user modifies any or all of the givens defined in Conditions and reruns the process
- if the user tells the PAA to implement a suggested solution, the PAA implements the solution, revalidates, replans if necessary, and reports results to the user

Scenario 3

Class: Baseline Development

Enabler: System Infrastructure, Intra-Flight Value Added, Inter-Flight Value Added

Goal: User definition of constraints; PAA definition of intra-flight plans, inter-flight plans, and validation of plans

Conditions: The user knows the following:

- the SSF configuration, including definition of assembly elements
- NSTS constraints, including standards for mass, volume, center of gravity (CG), Remote Manipulator System (RMS) utilization,

©ISX Corporation

and EVA
- major milestones, i.e., dates for milestones such as Man-Tended Capability (MTC), Permanent Manned Capability (PMC), and Assembly Complete (AC)
- dates and definition of phases within major milestones, e.g., Spacecraft Activation, Primary Hardware Control, and International Element Delivery
- priorities for each phase, e.g., spacecraft activation, health and status data to the ground, respond to command and control signals, and insure continuance of assembly -- all priorities for the Spacecraft Activation phase

Process: The PAA/user system performs the following:
- the user enters assembly elements, NSTS constraints, milestones, phases, priorities into the PAA (constraints)
- the PAA defines number and timing of flights and assigns assembly elements to flights
- the PAA defines cargo elements
- the PAA places cargo elements in the NSTS cargo bay for each flight
- the PAA computes values for measures including mass, volume, center of gravity (CG), Remote Manipulator System (RMS) utilization, EVA requirements, IVA requirements, operability, reliability.
- the PAA compares computed values to standards defined by constraints and identifies and assesses criticality of violated constraints
- the PAA identifies and prioritizes potential intra-flight or inter-flight solutions to constraint-violation problems
- the PAA develops task plans for implementing the solution
- the PAA reports violated constraints, estimated criticality of violations, and suggested solutions to the user
- the user tells the PAA to implement the high priority suggested solution OR the user modifies any or all of the givens defined in Conditions and reruns the process
- if the user tells the PAA to implement a suggested solution, the PAA implements the solution, revalidates, replans if necessary, and reports results to the user

5.2 What-If Scenarios

Scenario 4

Class: What-If

Enabler: System Infrastructure, Intra-Flight Value Added, Inter-Flight Value Added

Goal: User definition of a constraint change; PAA revision of constraints, revision of plans, and revalidation

Conditions: The following exists in the PAA:
- models of constraints, including NSTS constraints, SSF Hardware constraints, and programmatic constraints
- inter-flight and intra-flight assembly sequence plans consistent with constraints or inconsistent in known ways

©ISX Corporation

The user provides the following:
- a constraint change, e.g., use of Advanced Solid Rocket Motors (ASRM)

Process: The PAA/user system performs the following:
- the user enters the constraint change into the PAA, e.g., use of the ASRM increases NSTS payload mass capacity from 47,000 to 60,000 pounds
- the PAA propagates the change to affected constraints
- the PAA revalidates the existing plan against changed constraints, finding for example that the current plan does not load to mass capacity
- the PAA identifies alternative solutions to the problem, e.g., reassign cargo elements to flights followed by relocation of cargo elements to flights with possible redefinition of cargo elements
- the PAA reports the violated constraint and the suggested solutions to the user
- the user tells the PAA to implement the suggested solution
- the PAA implements the suggested solution
- the PAA revalidates the changed plan, develops solutions to new problems if necessary, and reports the results to the user

Scenario 5

Class: What-If

Enabler: System Infrastructure, Intra-Flight Value Added, Inter-Flight Value Added

Goal: User definition of a plan change; PAA revision of constraints, revision of plans, and revalidation

Conditions: The following exists in the PAA:
- models of constraints, including NSTS constraints, SSF Hardware constraints, and programmatic constraints
- inter-flight and intra-flight assembly sequence plans consistent with constraints or inconsistent in known ways

The user provides the following:
- a plan change, e.g., a reassignment of two cargo elements to different flights

Process: The PAA/user system performs the following:
- the user enters the plan change into the PAA, e.g., the cargo elements are moved to different flights
- the PAA propagates the change to affected plan elements
- the PAA revalidates the changed plan, finding for example that the new plan violates the CG constraint
- the PAA identifies alternative solutions to the problem, e.g., relocate cargo elements in the cargo bay or reassign cargo elements or redefine cargo elements
- the PAA reports the violated constraint and the suggested solutions to the user
- the user tells the PAA to implement the suggested solution
- the PAA implements the suggested solution
- the PAA revalidates the changed plan, develops solutions to new problems if necessary, and reports the results to the user

©ISX Corporation

6.0 Object Details

This section provides detailed object descriptions, including structural relationships, attributes, and processing. The focus is on the Personal Analysis Assistant; descriptions of other object classes identified in Section 4 are provided only to the extent necessary to support description of PAA objects and interfaces to PAA objects. The section begins with a description of PAA objects. This is followed by descriptions of objects in the environment external to the PAA -- the non-PAA objects introduced in Section 4. The section ends with descriptions of interfaces between PAA objects and external objects.

6.1 The PAA

6.1.1 Structure

PAA objects form an *assembly structure*; that is, objects are parts of other objects. The PAA assembly structure is summarized in Figure 6. The figure uses a triangle to indicate an "is part of" relationship, e.g., the hardware platform and the software applications are parts of the PAA.

6.1.1.1 Hardware Platform

The hardware platform object is defined below. The hardware platform has been specified by the customer to be a Macintosh II series platform. No further specification will be provided.

Object: Hardware Platform
Description: Hardware platform for System Infrastructure, Intra-Flight Value-Added, and Inter-Flight Value-Added software. Customer-specified to be Macintosh II series platforms widely available at the SSFPO and Booz-Allen & Hamilton facilities.

6.1.1.2 Software Applications

Top-level software application objects are defined below. Each top-level PAA software object (System Infrastructure, Intra-Flight Value Added, and Inter-Flight Value Added) are decomposed as shown in Figure 6. Descriptions of these lower-level objects are presented in Section 6.1.2.

Object: System Infrastructure
Description: Software that provides the capability for building representations of assembly sequence plans, e.g., flights, manifests, cargo elements, and associated performance measures; networks of dependencies among plan objects; constraint networks describing bounds on measures; and user-specifications of constraints and analysis options. It also supports calculation of performance measures and identification of violated constraints, and intra-flight plans. The PAA System Infrastructure is built on the MAX framework, which supports development of objects described in Section 6.1.2 and extension of associated knowledge bases by SSFPO and Booz-Allen & Hamilton staff.

Object: Intra-Flight Value-Added
Description: Software that, given the manifest for each flight, defines cargo elements, places them in the NSTS cargo bay, computes performance measure values, and identifies violated constraints. PAA Intra-Flight

©ISX Corporation

Value-Added is built on the MAX framework, which supports development of objects described in Section 6.1.2 and extension of associated knowledge bases by SSFPO and Booz-Allen & Hamilton staff.

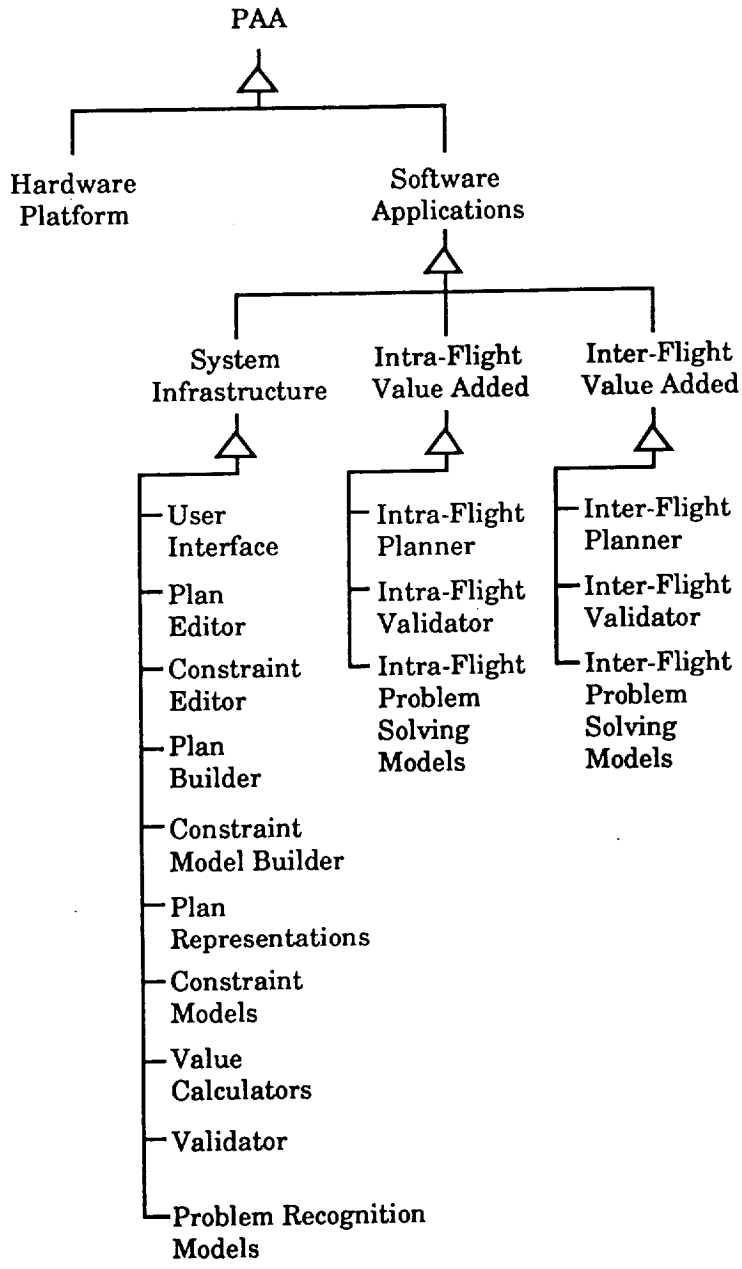


Figure 6: Personal Analysis Assistant Structure

Object: Inter-Flight Value-Added
Description: Software that, given major milestone dates and capability requirements, determines the number and dates of required flights and

©ISX Corporation

develops a manifest for each flight. PAA Inter-Flight Value-Added is built on the MAX framework, which supports development of objects described in Section 6.1.2 and extension of associated knowledge bases by SSFPO and Booz-Allen & Hamilton staff.

6.1.2 Processing

Figure 7 shows lower-level PAA software application objects and message connections. The Assembly Sequence Planners object is also shown to provide context. Other objects external to the PAA and the interfaces of these objects with PAA objects are described in Sections 6.2 and 6.3, respectively.

The figure is intended to be an overview of the behavior of the system, providing context for understanding the isolated behaviors of the objects described below. The figure is divided into three parts: System Infrastructure objects, Intra-Flight Value Added objects, and Inter-Flight Value Added objects. The Problem Solving Model object is shown overlapping the boundaries of the Intra-Flight Value Added and Inter-Flight Value Added top-level objects because the problem solving knowledge in the Problem Solving Model object is applicable to both the Intra-Flight Planner and the Inter-Flight Planner.

6.1.2.1 User Interface Specification

Object Name: User-Interface

Description: System Infrastructure software that provides "user friendly" interfaces to plan and constraint editors, representations of plan and constraint models, and the Validator object. The User-Interface object also provides input checking, a display manager, and maintenance of a User Model that enables interpretation of user input and preparation by the display manager of output meeting the user's information requirements.

Attributes: Selection <type, location>
User-Input <type, completeness, consistency>
System-Input <type, completeness, consistency, level of abstraction>
Information-Requirements <type, level of abstraction>
User-Output <type, location, level-of-abstraction>

Processing: Check-Selection
- check selection arguments against selection attribute constraints
- if constraints not met, return <no match> to display-manager
- if display-manager receives a <no match>, it prepares and posts a selection-failure message on the display
- if constraints are met, pass through selection to appropriate receiver

Check-User-Input
- check input arguments against input attribute constraints
- if a violation occurs, return violation identity to display-manager
- check input arguments against expectation defined by user-model
- if a consistency violation occurs, return violation identity to display-manager
- if display-manager receives a violation identity, it prepares and posts an input-violation message on the display
- if constraints are met, update information requirements
- if information requirements are updated, pass input and information requirements to appropriate receiver

©ISX Corporation

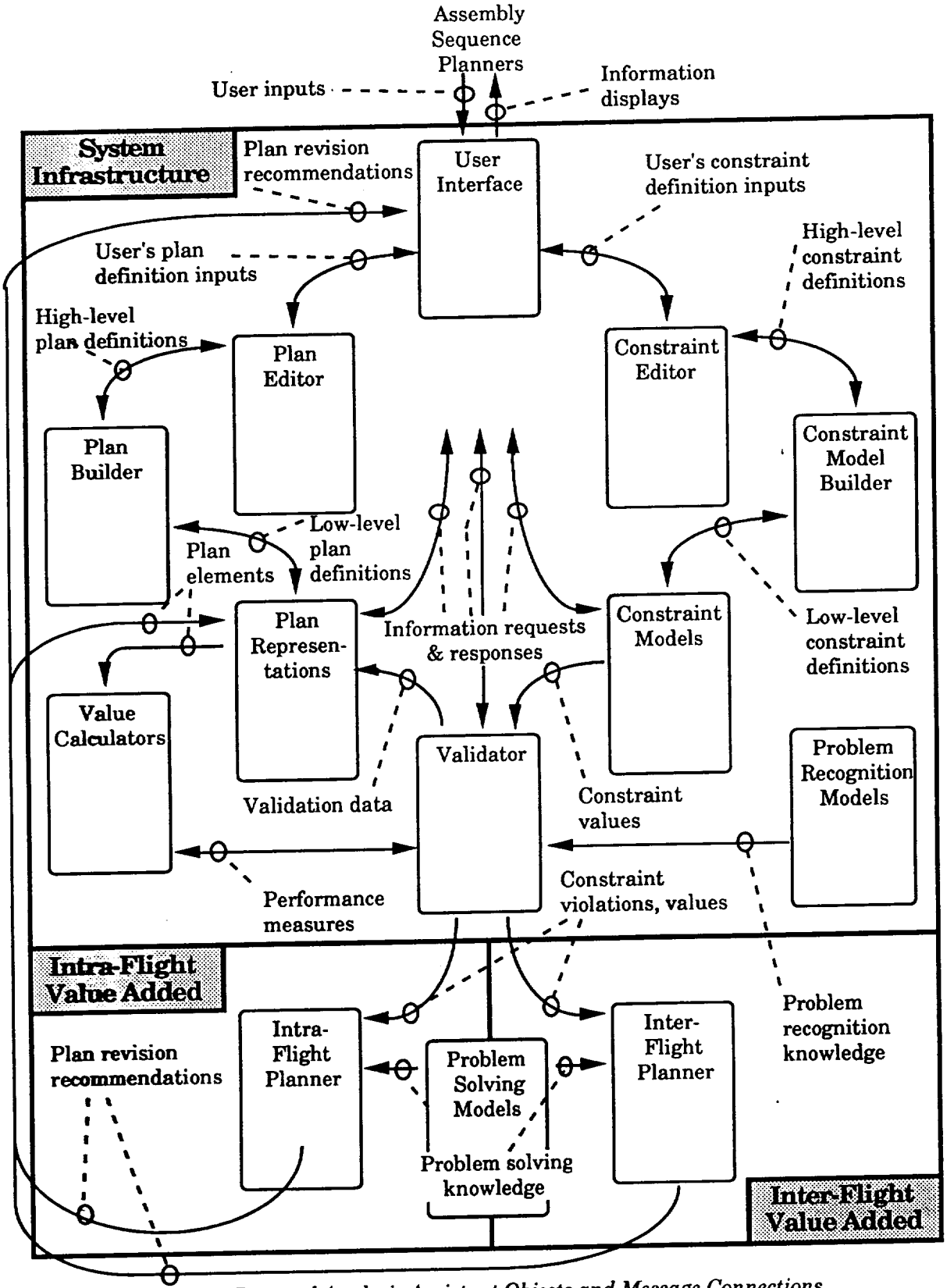


Figure 7: Personal Analysis Assistant Objects and Message Connections

©ISX Corporation

Update-User-Model
- with user-input and expectation from user-model, define new expectation
- send new expectation to user-model

Update-Information-Requirements
- with user-input and expectation from user-model, define type and level of abstraction of information required

Prepare-Output
- check system-input against information requirements
- if a violation occurs, return system-input and information-requirements to sender
- if no violation occurs, send system-input to display-manager
- if display-manager receives system-input, it prepares and posts information on the display

6.1.2.2 Plan Editor Specification

Object Name: Plan-Editor

Description: System Infrastructure software that provides capabilities supporting viewing, selecting, and editing assembly sequence plan elements, including the launch manifest, cargo elements, placement of cargo elements in the NSTS cargo bay, and assembly elements.

Attributes:
Selection <type>
User-Input <type>
System-Input <element-type, completeness, consistency, level of abstraction>
Add-Element <element-type>
Change-Element <element-type, change-type>
Delete-Element <element-type>
Plan-Builder-Output <type>

Processing:
Check-Selection
- check selection arguments against selection attribute constraints
- if constraints not met, return <no match> to User-Interface object
- if constraints are met, continue to check-system-input

Check-System-Input
- check system-input arguments against system-input attribute constraints
- if a violation occurs, return violation identity to sender

Add-Plan-Element
- check user-input and system-input arguments against add-element attribute constraints
- if a violation occurs, return violation identity to sender
- if no violation occurs, send input to Plan-Builder object
- send process completion message to requestor

Change-Plan-Element
- check user-input and system-input arguments against change-element attribute constraints
- if a violation occurs, return violation identity to sender
- if no violation occurs, send input to Plan-Builder object
- send process completion message to requestor

Delete-Plan-Element
- check user-input and system-input arguments against delete-element attribute constraints

©ISX Corporation

- if a violation occurs, return violation identity to sender
- if no violation occurs, send input to Plan-Builder object
- send process completion message to requestor

6.1.2.3 Constraint Editor Specification

Object Name: Constraint-Editor
Description: System Infrastructure software that provides capabilities supporting viewing, selecting, and editing constraint models, including NSTS Constraints, SSF Hardware Constraints, and Programmatic Constraints.
Attributes: Selection <type>
User-Input <type>
System-Input <type, completeness, consistency, level of abstraction>
Add-Constraint <constraint-type>
Change-Constraint <constraint-type>
Delete-Constraint <constraint-type>
Constraint-Model-Builder-Output <type>
Processing: Check-Selection
- check selection arguments against selection attribute constraints
- if selection attribute constraints not met, return <no match> to User-Interface object
- if constraints are met, continue to check-system-input

Check-System-Input
- check system-input arguments against system-input attribute constraints
- if a system-input attribute constraint violation occurs, return violation identity to sender
- if no violation occurs, pass input to appropriate procedure

Add-Constraint
- check user-input and system-input arguments against add-constraint attribute constraints
- if a violation occurs, return violation identity to sender
- if no violation occurs, send input to Plan Builder
- send process completion message to requestor

Change-Constraint
- check user-input and system-input arguments against change-constraint attribute constraints
- if a violation occurs, return violation identity to sender
- if no violation occurs, send input to Constraint-Model-Builder
- send process completion message to requestor

Delete-Constraint
- check user-input and system-input arguments against delete-constraint attribute constraints
- if a violation occurs, return violation identity to sender
- if no violation occurs, send input to Constraint-Model-Builder
- send process completion message to requestor

6.1.2.4 Plan Builder Specification

Object Name: Plan-Builder
Description: System Infrastructure software that, given high-level Plan-Editor requests to add, change, or delete plan elements (assembly elements,

©ISX Corporation

cargo elements, cargo bay placement, flights, and launch manifests), generates the implementation of requests in the Plan-Representation. Implementation of plan element edits includes generation of low-level effects from high-level edits and propagation of effects throughout the affected Plan-Representation.

Attributes: Selection <type>
System-Input <type, completeness, consistency, level of abstraction>
Add-Element-Problem <element-type, problem-type, criticality>
Change-Element-Problem <element-type, change-type, problem-type, criticality>
Delete-Element-Problem <element-type, problem-type, criticality>

Processing:

Check-Selection
- check selection arguments against selection attribute constraints
- if constraints not met, return <no match> to Plan-Editor object
- if constraints are met, continue to check-system-input

Check-System-Input
- check system-input arguments against system-input attribute constraints
- if a violation occurs, return violation identity to sender
- if no violation occurs, pass input to appropriate procedure

Add-Plan-Element
- identify affected Plan-Representation context
- identify proposed tasks for implementation of plan element addition
- identity problems created by proposed tasks for implementation of plan element addition (primary problems)
- identify secondary tasks created by proposed tasks for implementation of plan element addition (propagation tasks)
- identity problems created by secondary tasks for implementation of plan element addition (secondary problems)
- check primary and secondary problems against add-element-problem attribute constraints
- if a violation occurs, return violation identity and criticality to sender
- if no violation occurs, implement all tasks
- if sender confirms change, implement all tasks even with violations
- send process completion message to requestor

Change-Plan-Element
- identify affected Plan-Representation context
- identify proposed tasks for implementation of plan element change
- identity problems created by proposed tasks for implementation of plan element change (primary problems)
- identify secondary tasks created by proposed tasks for implementation of plan element change (propagation tasks)
- identity problems created by secondary tasks for implementation of plan element change (secondary problems)
- check primary and secondary problems against change-element-problem attribute constraints
- if a violation occurs, return violation identity and criticality to sender
- if no violation occurs, implement all tasks

©ISX Corporation

- if sender confirms change, implement all tasks even with violations
- send process completion message to requestor

Delete-Plan-Element
- identify affected Plan-Representation context
- identify proposed tasks for implementation of plan element deletion
- identity problems created by proposed tasks for implementation of plan element deletion (primary problems)
- identify secondary tasks created by proposed tasks for implementation of plan element deletion (propagation tasks)
- identity problems created by secondary tasks for implementation of plan element deletion (secondary problems)
- check primary and secondary problems against delete-element-problem attribute constraints
- if a violation occurs, return violation identity and criticality to sender
- if no violation occurs, implement all tasks
- if a violation occurs and sender confirms deletion, implement all tasks
- identity problems created by implemented tasks (primary and secondary problems)
- check primary and secondary problems against delete-element-problem attribute constraints
- if a violation occurs, return violation identity and criticality to sender
- send process completion message to requestor

6.1.2.5 Constraint Model Builder Specification

Object Name: Constraint-Model-Builder

Description: System Infrastructure software that, given high-level Constraint-Editor requests to add, change, or delete constraints (NSTS Constraints, SSF Hardware Constraints, Programmatic Constraints), generates the implementation of requests in the Constraint-Model. Implementation of constraint edits includes generation of low-level effects from high-level edits and propagation of effects throughout the affected Constraint-Model.

Attributes: Selection <type>
System-Input <type, completeness, consistency, level of abstraction>
Add-Constraint-Problem <constraint-type, problem-type, criticality>
Change-Constraint-Problem <constraint-type, problem-type, criticality>
Delete-Constraint-Problem <constraint-type, problem-type, criticality>

Processing: Check-Selection
- check selection arguments against selection attribute constraints
- if constraints not met, return <no match> to sender
- if constraints are met, continue to check-system-input

Check-System-Input
- check system-input arguments against system-input attribute constraints
- if a violation occurs, return violation identity to sender
- if no violation occurs, pass input to appropriate procedure

Add-Constraint

©ISX Corporation

- identify affected Constraint-Model context
- identify proposed tasks for implementation of constraint addition
- identity problems created by proposed tasks for implementation of constraint addition (primary problems)
- identify secondary tasks created by proposed tasks for implementation of constraint addition (propagation tasks)
- identity problems created by secondary tasks for implementation of constraint addition (secondary problems)
- check primary and secondary problems against add-constraint-problem attribute constraints
- if a violation occurs, return violation identityidentity and criticality to sender
- if no violation occurs, implement all tasks
- if a violation occurs and sender confirms addition, implement all tasks
- identity problems created by implemented tasks (primary and secondary problems)
- check primary and secondary problems against add-element-problem attribute constraints
- if a violation occurs, return violation identity and criticality to sender
- send process completion message to requestor

Change-Constraint

- identify affected Constraint-Model context
- identify proposed tasks for implementation of constraint change
- identity problems created by proposed tasks for implementation of constraint change (primary problems)
- identify secondary tasks created by proposed tasks for implementation of constraint change (propagation tasks)
- identity problems created by secondary tasks for implementation of constraint change (secondary problems)
- check primary and secondary problems against change-element-problem attribute constraints
- if a violation occurs, return violation identity and criticality to sender
- if no violation occurs, implement all tasks
- if a violation occurs and sender confirms change, implement all tasks
- identity problems created by implemented tasks (primary and secondary problems)
- check primary and secondary problems against change-element-problem attribute constraints
- if a violation occurs, return violation identity and criticality to sender
- send process completion message to requestor

Delete-Constraint

- identify affected Constraint-Model context
- identify proposed tasks for implementation of constraint deletion
- identity problems created by proposed tasks for implementation of constraint deletion (primary problems)
- identify secondary tasks created by proposed tasks for implementation of constraint deletion (propagation tasks)
- identity problems created by secondary tasks for implementation

©ISX Corporation

of constraint deletion (secondary problems)
- check primary and secondary problems against delete-element-problem attribute constraints
- if a violation occurs, return violation identity and criticality to sender
- if no violation occurs, implement all tasks
- if a violation occurs and sender confirms deletion, implement all tasks
- identity problems created by implemented tasks (primary and secondary problems)
- check primary and secondary problems against delete-element-problem attribute constraints
- if a violation occurs, return violation identity and criticality to sender
- send process completion message to requestor

6.1.2.6 Plan Representation Specification

Object Name: Plan-Representation

Description: System Infrastructure software that stores assembly sequence plan representations, including assembly elements, cargo elements, cargo bay placements, flights, launch manifests, and interrelations among plan elements. Plan-Representation also stores the results of evaluations of plan representations produced by the Validator and provides information to the User-Interface, the Plan-Builder, and Value-Calculators. Changes in plan representations are accomplished through messages sent by the Plan-Builder, the Intra-Flight-Planner, or the Inter-Flight-Planner.

Attributes: System-Input <type, completeness, information requirements>
Add-Element-Problem <element-type, problem-type, criticality>
Change-Element-Problem <element-type, change-type, problem-type, criticality>
Delete-Element-Problem <type, problem-type, criticality>
Value-Calculation <type, plan-representation-identifier>

Processing: Check-System-Input
- check system-input arguments against system-input attribute constraints
- if a violation occurs, return violation identity to sender
- if no violation occurs, pass input to appropriate procedure

Add-Plan-Element
- implement tasks for plan element addition
- identity problems created by implementation of plan element addition
- check problems against add-element-problem attribute constraints
- if a violation occurs, return violation identity and criticality to sender
- send process completion message to requestor

Change-Plan-Element
- implement tasks for plan element change
- identity problems created by implementation of plan element change
- check problems against change-element-problem attribute

©ISX Corporation

constraints
- if a violation occurs, return violation identity and criticality to sender
- send process completion message to requestor

Delete-Plan-Element
- implement tasks for plan element deletion
- identity problems created by implementation of plan element deletion
- check problems against delete-element-problem attribute constraints
- if a violation occurs, return violation identity and criticality to sender
- send process completion message to requestor

Request-Value-Calculation
- form value-calculation request using value-calculation attribute
- send value-calculation request to Value-Calculation

Record-Validation
- associate validation data received from Validation with appropriate plan element

Information-Response
- identify information meeting system-input information requirements
- get requested information
- send information to requestor

6.1.2.7 Constraint Model Specification

Object Name: Constraint-Models

Description: System Infrastructure software that stores assembly sequence plan constraints, including NSTS Constraints, SSF Hardware Constraints, Programmatic Constraints, and interrelations among constraints. Constraint-Models also provides information to the User-Interface, the Constraint-Model-Builder, and the Validator. Changes in constraint models are accomplished through messages sent by the Constraint-Model-Builder.

Attributes: System-Input <type, completeness, information requirements>
Add-Constraint-Problem <constraint-type, problem-type, criticality>
Change-Constraint-Problem <constraint-type, change-type, problem-type, criticality>
Delete-Constraint-Problem <type, problem-type, criticality>

Processing: Check-System-Input
- check system-input arguments against system-input attribute constraints
- if a violation occurs, return violation identity to sender
- if no violation occurs, pass input to appropriate procedure

Add-Constraint
- implement tasks for constraint addition
- identity problems created by implementation of constraint addition
- check problems against add-element-problem attribute constraints
- if a violation occurs, return violation identity and criticality to sender

©ISX Corporation

- send process completion message to requestor

Change-Constraint
- implement tasks for constraint change
- identity problems created by implementation of constraint change
- check problems against change-element-problem attribute constraints
- if a violation occurs, return violation identity and criticality to sender
- send process completion message to requestor

Delete-Constraint
- implement tasks for constraint deletion
- identity problems created by implementation of constraint deletion
- check problems against delete-element-problem attribute constraints
- if a violation occurs, return violation identity and criticality to sender
- send process completion message to requestor

Information-Response
- identify information meeting system-input information requirements
- get requested information
- send information to requestor

6.1.2.8 Value Calculator Specification

Object Name: Value-Calculator

Description: System Infrastructure software that identifies and computes performance measures appropriate to the plan representation information sent by the Plan-Representation object. Measures include cargo mass, volume, center of gravity (CG), power requirements, extra-vehicular activity (EVA) requirement, and Remote Manipulator System (RMS) reach requirements. This object provides information to the Validator.

Attributes: System-Input <type, completeness>
Report-Form <type, parameters, plan-representation>

Processing: Check-System-Input
- check system-input arguments against system-input attribute constraints
- if a violation occurs, return violation identity to sender
- if no violation occurs, pass input to appropriate procedure

Identify-Measures
- select measures appropriate for plan representation contained in value-calculation-request received from Plan-Representations

Get-Plan-Representation
- request plan data from Plan-Representation

Compute-Measures
- compute identified measures on plan data supplied by Plan-Representation

Report-Measures
- form report using Report-Form attributes
- send computed performance measures with plan representation identifier to Validator

©ISX Corporation

6.1.2.9 Validator Specification

Object Name: Validator

Description: System Infrastructure software that compares values of performance measures provided by Value-Calculator to standards defined by Constraint-Models and determines whether measures violate constraints using knowledge provided by Problem-Recognition-Models. Provides information to the User-Interface, Plan-Representations, the Intra-Flight Planner and the Inter-Flight Planner.

Attributes: System-Input <type, completeness>
Report-Form <type, parameters, plan-representation>

Processing: Check-System-Input
- check system-input arguments against system-input attribute constraints
- if a violation occurs, return violation identity to sender
- if no violation occurs, pass input to appropriate procedure

Get-Constraints
- identify constraints relevant to perfomance measures provided by the Value-Calculator
- get identified constraints

Validate
- compare performance measures with standards defined by constraints
- send comparisons to Problem-Recognition-Models for evaluation

Report-Results
- develop report using response from Problem-Recognition-Models and Report-Form attributes
- send report with plan representation identifier to User-Interface, Plan-Representations, Intra-Flight-Planner, and Inter-Flight-Planner.

6.1.2.10 Problem Recognition Model Specification

Object Name: Problem-Recognition-Models

Description: System Infrastructure software that is a knowledge-based server for the Validator. The object evaluates comparisons of performance measures and standards defined by constraints provided by the Validator to determine whether measures violate constraints. It provides results of the evaluation to the Validator.

Attributes: System-Input <type, completeness>

Processing: Check-System-Input
- check system-input arguments against system-input attribute constraints
- if a violation occurs, return violation identity to sender
- if no violation occurs, pass input to appropriate procedure

Identify-Knowledge
- identify knowledge relevant to evaluating measure-constraint comparisons provided by the Validator

Evaluate
- execute evaluation by applying identified knowledge to measure-constraint comparisons
- send results to the Validator

©ISX Corporation

6.1.2.11 Intra-Flight Planner Specification

Object Name: Intra-Flight-Planner

Description: Intra-Flight Value Added software that, given intra-flight constraint violation problems identified by the Validator and problem solutions suggested by Intra-Flight-Problem-Solving-Models, selects and implements solutions. Solutions are implemented by developing and executing task plans producing required changes to the Plan-Representation. Provides information to the User-Interface and Plan-Representations.

Attributes: Selection <type>
System-Input <type, completeness, consistency, level of abstraction>
Solution-Request <problem-type>
Add-Element-Problem <element-type, problem-type, criticality>
Change-Element-Problem <element-type, change-type, problem-type, criticality>
Delete-Element-Problem <element-type, problem-type, criticality>

Processing: Check-System-Input
- check system-input arguments against system-input attribute constraints
- if a violation occurs, return violation identity to sender
- if no violation occurs, pass input to appropriate procedure

Get-Solutions
- prepare solution-request using solution-request attributes
- send solution-request to Problem-Solving-Models

Select-Solution
- evaluate solutions provided by Problem-Solving-Models to identify a recommended solution
- send recommendation to User-Interface
- if user confirms change, implement all tasks
- if user does not confirm change, select and recommend another solution
- if no solution can be selected, stop

Implement-Addition-Tasks
- identify affected Plan-Representation context
- identify proposed tasks for implementation of plan element addition
- identity problems created by proposed tasks for implementation of plan element addition (primary problems)
- identify secondary tasks created by proposed tasks for implementation of plan element addition (propagation tasks)
- identity problems created by secondary tasks for implementation of plan element addition (secondary problems)
- check primary and secondary problems against add-element-problem attribute constraints
- if a violation occurs, return violation identity and criticality to sender
- if no violation occurs, implement all tasks
- if sender confirms change, implement all tasks even with violations
- send process completion message to requestor

Implement-Change-Tasks

©ISX Corporation

- identify affected Plan-Representation context
- identify proposed tasks for implementation of plan element change
- identity problems created by proposed tasks for implementation of plan element change (primary problems)
- identify secondary tasks created by proposed tasks for implementation of plan element change (propagation tasks)
- identity problems created by secondary tasks for implementation of plan element change (secondary problems)
- check primary and secondary problems against change-element-problem attribute constraints
- if a violation occurs, return violation identity and criticality to sender
- if no violation occurs, implement all tasks
- if sender confirms change, implement all tasks even with violations
- send process completion message to requestor

Implement-Delete-Tasks

- identify affected Plan-Representation context
- identify proposed tasks for implementation of plan element deletion
- identity problems created by proposed tasks for implementation of plan element deletion (primary problems)
- identify secondary tasks created by proposed tasks for implementation of plan element deletion (propagation tasks)
- identity problems created by secondary tasks for implementation of plan element deletion (secondary problems)
- check primary and secondary problems against delete-element-problem attribute constraints
- if a violation occurs, return violation identity and criticality to sender
- if no violation occurs, implement all tasks
- if a violation occurs and sender confirms deletion, implement all tasks
- identity problems created by implemented tasks (primary and secondary problems)
- check primary and secondary problems against delete-element-problem attribute constraints
- if a violation occurs, return violation identity and criticality to sender
- send process completion message to requestor

6.1.2.12 Inter-Flight Planner Specification

Object Name: Inter-Flight-Planner
Description: Inter-Flight Value Added software that, given inter-flight constraint violation problems identified by the Validator and problem solutions suggested by Inter-Flight-Problem-Solving-Models, selects and implements solutions. Solutions are implemented by developing and executing task plans producing required changes to the Plan-Representation. Provides information to the User-Interface and Plan-Representations.
Attributes: Selection <type>

©ISX Corporation

System-Input <type, completeness, consistency, level of abstraction>
Solution-Request <problem-type>
Add-Element-Problem <element-type, problem-type, criticality>
Change-Element-Problem <element-type, change-type, problem-type, criticality>
Delete-Element-Problem <element-type, problem-type, criticality>

Processing:

Check-System-Input
- check system-input arguments against system-input attribute constraints
- if a violation occurs, return violation identity to sender
- if no violation occurs, pass input to appropriate procedure

Get-Solutions
- prepare solution-request using solution-request attributes
- send solution-request to Problem-Solving-Models

Select-Solution
- evaluate solutions provided by Problem-Solving-Models to identify a recommended solution
- send recommendation to User-Interface
- if user confirms change, implement all tasks
- if user does not confirm change, select and recommend another solution
- if no solution can be selected, stop

Implement-Addition-Tasks
- identify affected Plan-Representation context
- identify proposed tasks for implementation of plan element addition
- identity problems created by proposed tasks for implementation of plan element addition (primary problems)
- identify secondary tasks created by proposed tasks for implementation of plan element addition (propagation tasks)
- identity problems created by secondary tasks for implementation of plan element addition (secondary problems)
- check primary and secondary problems against add-element-problem attribute constraints
- if a violation occurs, return violation identity and criticality to sender
- if no violation occurs, implement all tasks
- if sender confirms change, implement all tasks even with violations
- send process completion message to requestor

Implement-Change-Tasks
- identify affected Plan-Representation context
- identify proposed tasks for implementation of plan element change
- identity problems created by proposed tasks for implementation of plan element change (primary problems)
- identify secondary tasks created by proposed tasks for implementation of plan element change (propagation tasks)
- identity problems created by secondary tasks for implementation of plan element change (secondary problems)
- check primary and secondary problems against change-element-problem attribute constraints
- if a violation occurs, return violation identity and criticality to sender

©ISX Corporation

- if no violation occurs, implement all tasks
- if sender confirms change, implement all tasks even with violations
- send process completion message to requestor

Implement-Delete-Tasks
- identify affected Plan-Representation context
- identify proposed tasks for implementation of plan element deletion
- identity problems created by proposed tasks for implementation of plan element deletion (primary problems)
- identify secondary tasks created by proposed tasks for implementation of plan element deletion (propagation tasks)
- identity problems created by secondary tasks for implementation of plan element deletion (secondary problems)
- check primary and secondary problems against delete-element-problem attribute constraints
- if a violation occurs, return violation identity and criticality to sender
- if no violation occurs, implement all tasks
- if a violation occurs and sender confirms deletion, implement all tasks
- identity problems created by implemented tasks (primary and secondary problems)
- check primary and secondary problems against delete-element-problem attribute constraints
- if a violation occurs, return violation identity and criticality to sender
- send process completion message to requestor

6.1.2.13 Intra-Flight Problem Solving Model Specification

Object Name: Problem-Solving-Models

Description: Intra-Flight and Inter-Flight Value Added software that is a knowledge-based server for the Intra-Flight Planner and the Inter-Flight Planner. Problem-Solving-Models contains knowledge about how to solve assembly sequence plan problems. The object uses information in the solution-request provided by the Intra-Flight or Inter-Flight Planner to identify and prioritize potential solutions. It provides results to the information requestor, either the Intra-Flight-Planner or the Inter-Flight-Planner.

Attributes: System-Input <type, completeness>

Processing: Check-System-Input
- check system-input arguments against system-input attribute constraints
- if a violation occurs, return violation identity to sender
- if no violation occurs, pass input to appropriate procedure

Identify-Knowledge
- identify knowledge relevant to identifying and prioritizing solutions
- execute identification and prioritization
- send results to requestor

©ISX Corporation

6.2 The External Environment

Objects forming the external environment of the PAA were introduced in Section 4. This section describes the structure of these objects to a level of detail permitting definition of the content of the PAA Constraint Model and Plan Representations objects and identification of interfaces between PAA objects and external information sources and information consumers. These external interfaces are defined in Section 6.3.

External environment objects form either an *assembly structure*like that described for PAA objects in Section 6.1 or a *hierarchical classification structure* in which objects are instances of other objects. A typical hierachical classification structure is shown in Figure 8 for the Information Sources object. The figure uses a semi-circle to indicate an "is a" relationship, e.g., an external system is an information source.

6.2.1 Assembly Sequence Planners

Although there is a hierarchical classification of Assembly Sequence Planners, e.g., levels of SSFPO planners and Booz-Allen & Hamilton planners, it is not necessary to define this structure in order to define PAA interfaces.

6.2.2 Information Sources

There are two information sources. The structure is shown in Figure 8. Descriptions of the two top-level objects follow the figure.

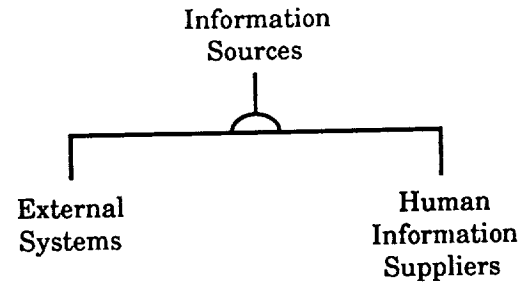


Figure 8: Information Sources Structure

Object: External Systems
Description: Databases and computational tools that provide information needed to generate information required by Information Consumers.

Object: Human Information Suppliers
Description: Persons, usually representatives of SSF functional organizations, who define constraints and other information needed to generate information requested by Information Consumers.

There are hierarchical classification structures below the External Systems object and the Human Information Suppliers object, and it may be necessary to decompose these objects to at least one more level in order to completely specify the PAA system. Further decomposition is not possible at the present time, however, because information on SSFPO external systems and NASA organizations is not currently available. Structural details for Information Sources will be specified in later versions of the requirements document, which will be developed in subsequent phases of PAA system development.

©ISX Corporation

6.2.3 Constraints

Real-world assembly sequence planning constraints will be represented in the PAA's Constraint Model object. There is a rich constraint structure for assembly sequence planning constraints, characterized by numerous and complex interrelationships among constraints. The first two levels of the hierarchical classification structure of constraints are shown in Figure 9. Further decomposition will be required, and the structure must be represented as a network in order to capture the complex relationships among constraints required for development of the Constraint Model object. The information required for this detailed analysis of constraints is not presently available to ISX, however, and will require extensive knowledge acquisition involving SSFPO and Booz-Allen & Hamilton staff early in the next phase of PAA system development.

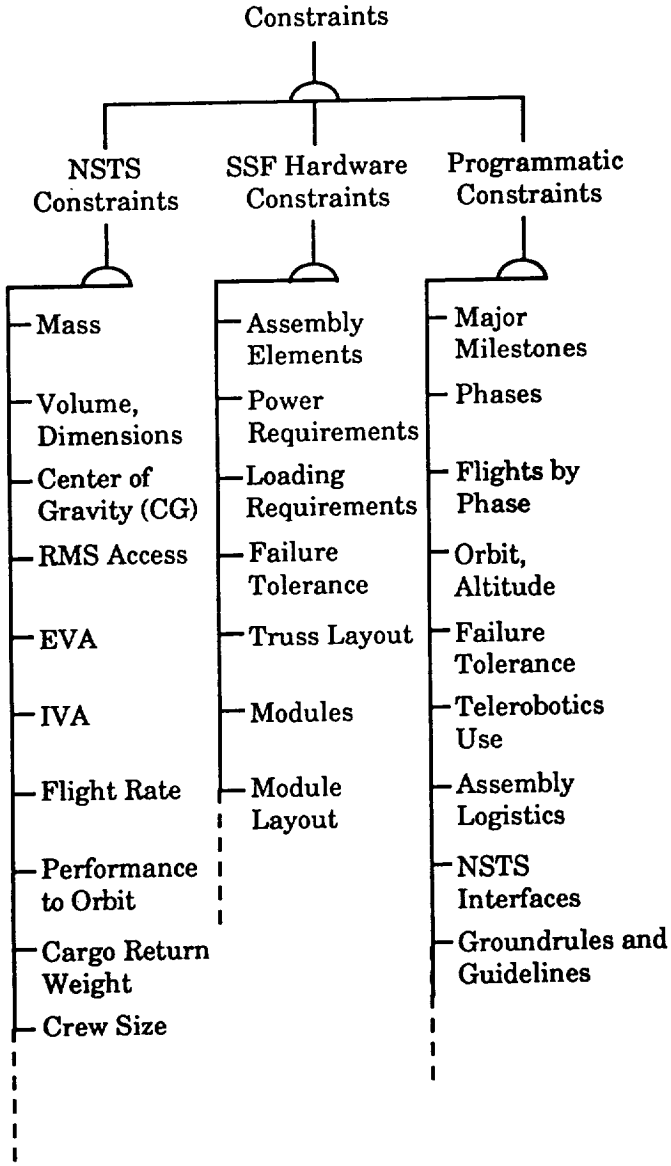


Figure 9: Constraints Structure

©ISX Corporation

First-level constraint objects are briefly described as follows:

Object: NSTS Constraints
Description: Constraints associated with the NSTS, e.g., flight rate, EVA support, and cargo bay mass, volume, center of gravity.

Object: SSF Hardware Constraints
Description: Constraints associated with SSF hardware configuration, e.g., assembly elements, power requirements, and mass, volume and shape of assembly elements.

Object: Programmatic Constraints
Description: Constraints defined by SSFPO and above, e.g., major milestones, system redundancy, life support.

6.2.4 Plan Elements

Assembly sequence plan elements will be represented in the Plan Representations object of the PAA. These representations will be manipulated by assembly sequence planners through the Plan Editor and by the Intra-Flight Planner and the Inter-Flight Planner. There are five top-level plan elements related in an assembly structure: the Launch Manifest, Flights, Cargo Elements, Cargo Element Locations, and Assembly Elements. Figure 10 shows the assembly structure for these objects, along with related objects: the cargo bay and validations and values for plan elements. The Cargo Bay object is represented in the PAA Constraint Model. The Validations and Values are produced by the PAA Validation and Value Calculator objects, respectively, and are stored with the plan representation.

Object: Launch Manifest
Description: The inter-flight assembly sequence plan which specifies the number and timing of flights, the assignment of cargo elements to flights, and the location of cargo elements in the NSTS cargo bay; the inter-flight plan.

Object: Flight
Description: Specification of the assignment of cargo elements to flights, the placement of cargo elements in the NSTS cargo bay; the intra-flight plan.

Object: Cargo Element
Description: Elements to be loaded into the NSTS cargo bay; may be an assembly element or an aggregation of assembly elements.

Object: Cargo Element Location
Description: The location in the NSTS cargo bay at which a cargo element is placed and the orientation of the cargo element. Related objects include treadle positions.

Object: Assembly Element
Description: Lowest level elements of the SSF configuration. Defined by SSF Hardware constraints.

©ISX Corporation

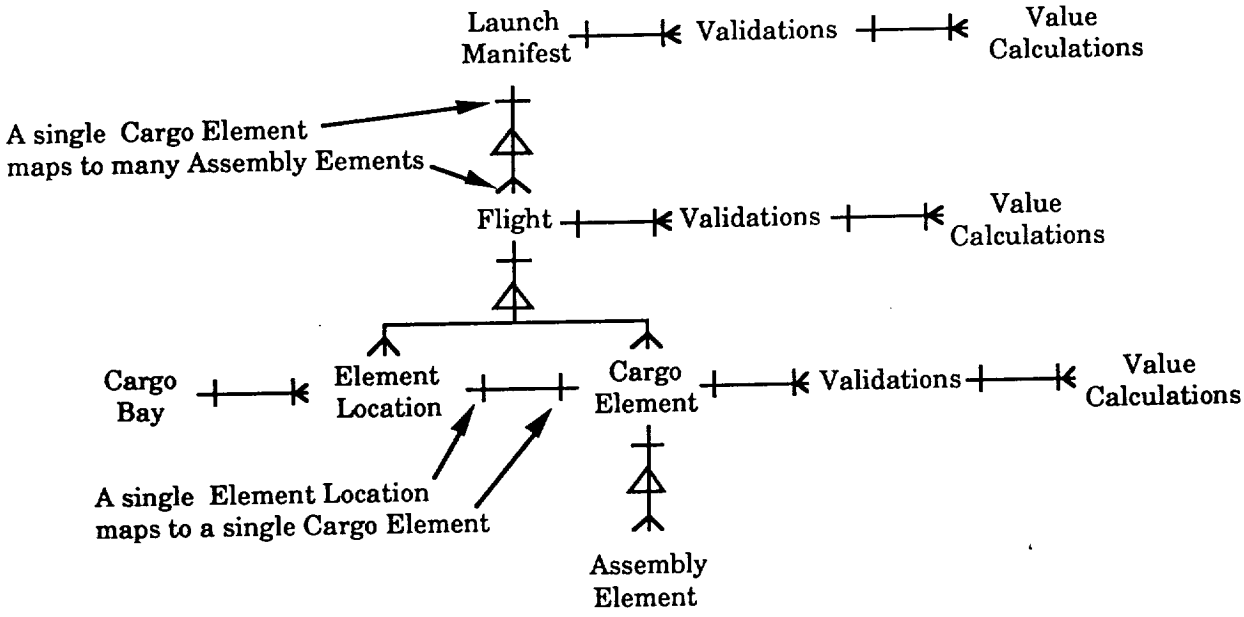


Figure 15: Structure of Plan Elements

6.2.5 Reports

As shown in Figure 16, there are two top-level reports. Descriptions of the two report objects follow the figure. Each report can be decomposed into assembly structures, but it is not necessary to define these structures in order to define PAA requirements.

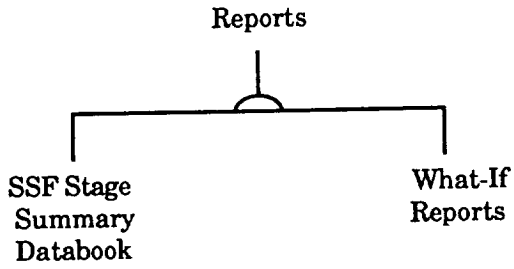


Figure 15: Reports Structure

Object: Space Station Stage Summary Databook

Description: A document issued periodically that provides information to SSF Information Consumers with information relative to the baseline assembly sequence and preliminary stage definition. It serves as a common database for lower level analyses, a mechanism to control the next level of detail with respect to the configuration assembly sequence, and a starting point for the stage definition process.

©ISX Corporation

Object: What-If Reports
Description: Documents issued in response to questions posed by Information Consumers regarding proposed perturbations of the baseline assembly sequence and preliminary stage definition documented in the Space Station Stage Summary Databook. The form of these documents is variable, ranging from formal reports to memos to presentations to telephone conversations.

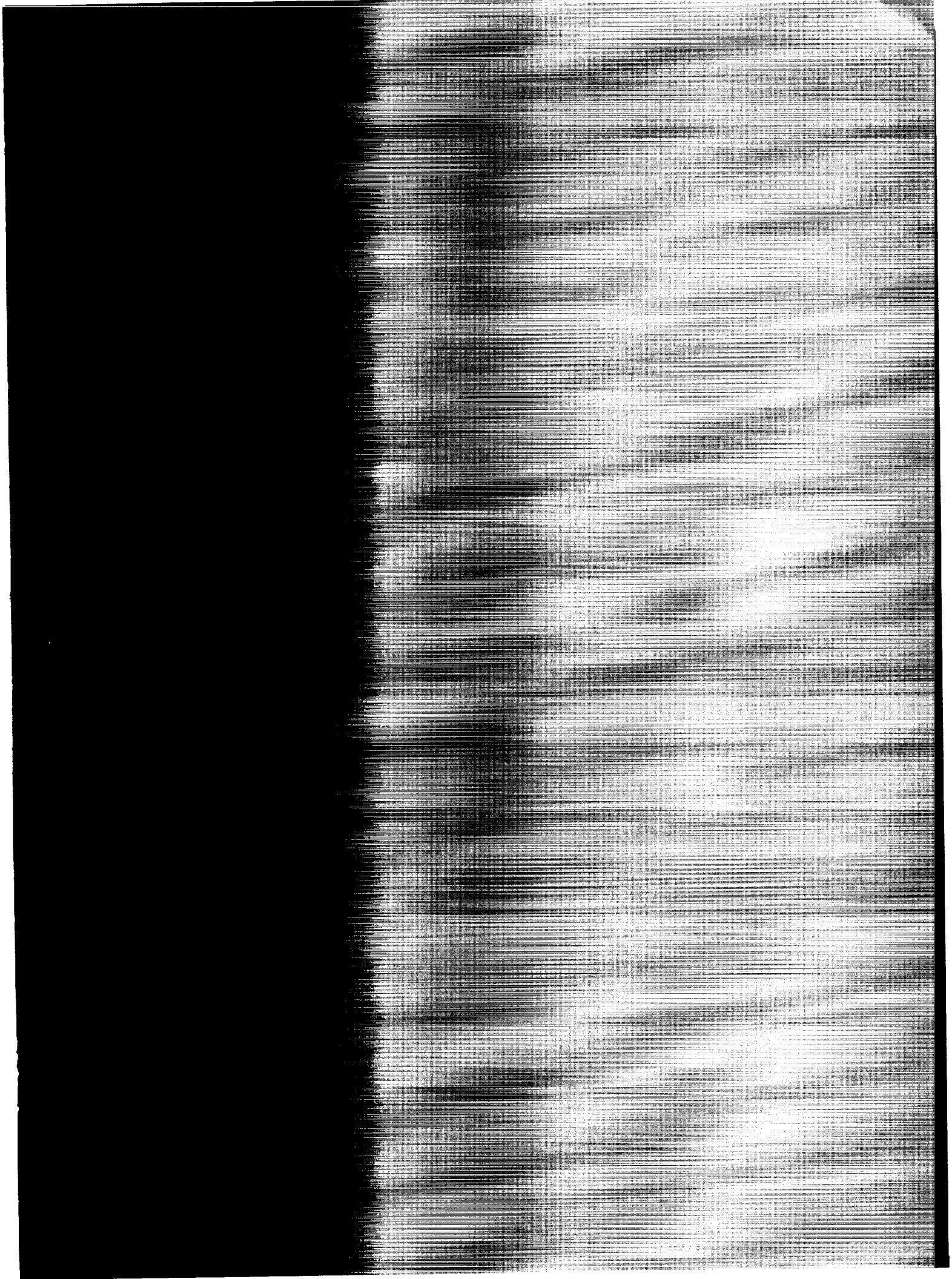
6.2.6 Information Consumers

There is a hierarchical classification structure of Information Consumers, but it is not necessary at the present time to define this structure below the class level in order to define PAA requirements.

6.3 External Interfaces

There are potentially interfaces between PAA objects and Information Sources and between PAA objects and Information Consumers. These interfaces cannot be defined at the present time because details of Information Sources and Information Consumers objects are not available to ISX. Because the customer has directed that there shall be no attempt to interface directly between the PAA and Information Sources or Information Consumers, no further specification of external interfaces will be provided.

©ISX Corporation



Preliminary Design Document

Knowledge-Based Decision-Support System for SSF Engineering Managers

Submitted By

ISX Corporation
501 Marin St., Suite 214
Thousand Oaks, CA 91360

June 15, 1990

Funded By

NASA Ames Research Center
Small Business Innovative Research Contract NAS2-13161

Table of Contents

1.0 Introduction

1.1 Purpose and Scope

This document describes the top-level system design for a knowledge-based decision-support system for Space Station Freedom (SSF) engineering managers. The target application is SSF assembly sequence planning. Development of this design is funded by NASA Ames Research Center (ARC) Phase I SBIR (Small Business Innovative Research) Contract NAS2-13161.

1.2 Identification

This is the first released version of the top-level system design document for a knowledge-based decision-support system for SSF engineering managers. It is designated version 1 and is dated June 15, 1990.

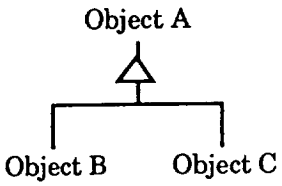
This work was performed using ISX's Intelligent Systems Engineering (ISE) methodology. A key feature of the ISE methodology is the early and continuous use of prototypes to inform the development of requirements and the system design. The top-level design described in this document, and the requirements specification cited in Section 2, were developed with the support of prototypes used to express and test assumptions and approaches. As additional prototypes are developed, requirements and the design will also continue to develop, increasing in both depth and breadth. These enhancements and extensions will be documented in subsequent drafts of this design document and the requirements specification, which will be issued periodically as the products of work performed under a subsequent contract.

1.3 Acronyms

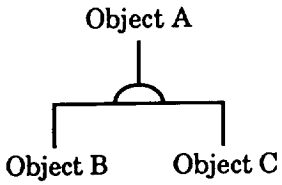
ARC	NASA's Ames Research Center
CG	Center of Gravity
FOA	Focus of Attention
ISE	Intelligent Systems Engineering
MAX	Manager's Assistant
NSTS	National Space Transportation System (the Shuttle)
PAA	Personal Analysis Assistant
SBIR	Small Business Innovative Research
SSF	Space Station Freedom
SSFPO	Space Station Freedom Program Office

©ISX Corporation

1.4 Notation



Assembly Structure: Object B *is part of* Object A



Hierarchical Classification Structure: Object B *is an* Object A

Instance Connection: 1 Object A is connected to many Object Bs

©ISX Corporation

1.5 Background

This work is funded by ARC Phase I SBIR Contract NAS2-13161. The objective is the development of specifications for a knowledge-based decision-support system for Space Station Freedom (SSF) engineering managers.

Following meetings with Paul Neumann and Ben Barker of the Space Station Freedom Program Office (SSFPO) and conversations with Henry Lum of ARC and Gregg Swietek of SSFPO, it was determined that SSF assembly sequence planning is an appropriate application for the Phase I SBIR, and that the SSFPO is sufficiently interested to provide staff to support application assessment, requirements analysis, and design activities. The customer and expert for the proposed application is Bill Bastedo of the SSFPO. Mr. Bastedo offered two days of his and his staff's time to support an application assessment, followed by occasional review of requirements and design documentation.

Based on experience derived from hundreds of projects conducted over the past seven years, ISX has developed an Intelligent Systems Engineering (ISE) methodology that combines the methods of systems engineering and knowledge engineering to meet the special systems development requirements posed by *intelligent systems*, systems that blend artificial intelligence and other advanced technologies with more conventional computing technologies. The ISE methodology defines a phased program process that begins with application screenings designed to provide a preliminary determination of the relative technical risks and payoffs associated with a potential application, and then moves through application assessments to requirements analysis, system design, and development.

The Application Assessment. Using information provided by Paul Neumann and Ben Barker, ISX completed the application screening in December, 1989. The next step, performed on February 26 and 27, 1990, was the application assessment. The assessment team included Bill Bewley, Gary Edwards, David Rosenberg, and Allen Smith of ISX. Peter Warren and Brook Sullivan of Booz-Allen & Hamilton were the application experts, and Bill Bastedo of the SSFPO provided application information in the form of feedback to a presentation of preliminary assessment results on the afternoon of February 27. The assessment indicated the value and feasibility of a personal "assistant" in a machine that would perform the work that is currently so tedious and time-consuming for the human assembly sequence planner. The assistant would be a "Personal Analysis Assistant" that helps the human planner by doing the bookkeeping to maintain plan data and executing the procedures and heuristics currently used by the human planner to define flights, develop flight manifests, define and place cargo elements, calculate performance measures, and identify violated constraints. The document "Application Assessment Report: Space Station Assembly Sequence Planning," cited in Section 2, presents assessment results in detail.

The Requirements Analysis. The requirements analysis for the Personal Analysis Assistant followed the application assessment. The document "Requirements Analysis: Knowledge-Based Decision-Support System for SSF Engineering Managers," cited in Section 2, describes the results of the requirements analysis. These results can be briefly summarized as follows. The Personal Analysis Assistant (PAA) is a set of software applications running on Macintosh II series platform as directed by SSFPO. The software applications are represented as three top-level objects: the System Infrastructure, Intra-Flight Value Added, and Inter-Flight Value Added. SSFPO and Booz-Allen & Hamilton staff will be able to extend knowledge bases associated with each object. The *System Infrastructure* is the substrate on which software elements providing inter-flight and intra-flight value-added functionality are built. It provides the capability for building representations of assembly sequence plans, e.g., flights, manifests, cargo elements, and

©ISX Corporation

associated performance measures; networks of dependencies among plan objects; constraint networks describing bounds on measures; and user-specifications of constraints and analysis options. It also supports calculation of performance measures and identification of violated constraints, and intra-flight plans. It is composed of the following lower-level objects: user interface, plan editor, constraint editor, plan builder, constraint model builder, plan representations, constraint models, value calculators, validator, and problem recognition models. *Intra-Flight Value-Added* provides functionality that will, given the manifest for each flight, define cargo elements, place them in the NSTS cargo bay, compute performance measure values, and identify violated constraints. Lower-level objects include the intra-flight planner, the intra-flight validator, and intra-flight problem-solving models. *Inter-Flight Value-Added* provides functionality that will, given major milestone dates and capability requirements, determine the number and dates of required flights and develop a manifest for each flight. Lower-level objects include the inter-flight planner, the inter-flight validator, and inter-flight problem-solving models. The definitions of these objects presented in Section 6 of "Requirements Analysis: Knowledge-Based Decision-Support System for SSF Engineering Managers" serve as a starting point for the more detailed definitions produced in the system design.

System Design. The ISE system design process begins with an examination of the major problem-solving tasks performed by the system as defined by the requirements analysis objects. These tasks are then collected into functionally-related groups, and the tasks and groupings are evaluated by applying them to the operational scenarios defined in the requirements analysis. Following revisions of task definitions and groupings that may be suggested by the evaluation, and possibly a series of subsequent evaluation-revision cycles, major system functions are identified, including the user interface. Alternative technologies are then assessed for their applicability to each system function, and "best fit" technologies are selected and assigned to functions to form major system components. Specification of system components and their interactions follows, and the resulting design is evaluated by mapping scenario events to system components. Design revisions suggested by the scenario-based evaluation may lead to a series of evaluation-revision cycles, usually based on a series of design prototypes. The design prototypes are also used to guide development of the user interface, each prototype expressing the current conception of the appearance and behavior of the system in interaction with the user. Evaluations of designs (and requirements) involve the user as a major contributor from the beginning of the process.

Task definitions, groupings, and their application to operational scenarios are described in Section 3. Section 4 presents current definitions of major system functions, including the user interface. Section 5 summarizes technology assessment, selection, and assignment to system functions. Detailed specifications of system components and their interactions are presented in Section 6.

2.0 References

Application Assessment Report: Space Station Assembly Sequence Planning. ISX Corporation. March 6, 1990.

Kaidy, James T., and Bastedo, William G. Space Station Assembly Sequence Planning: An engineering and operational challenge. *Proceedings of the AIAA*, 1988.

Requirements Specification: Knowledge-Based Decision-Support System for SSF Engineering Managers. Version 0.1, ISX Corporation, April 27, 1990.

©ISX Corporation

Space Station Stage Summary Databook. Space Station Freedom Program Office, December 15, 1989.

Warren, P. *Baseline Assembly Sequence Rationale*, February 28, 1990.

Warren, P. and Sullivan, B. *ASRM Trade Study*, 1990.

3.0 Task Definitions and Groupings

This section presents definitions of the major problem solving tasks to be performed by the PAA system and a description of functionally-related task groupings.

3.1 Major Problem Solving Tasks

As described in the requirements specification (see the reference in Section 2), assembly sequence planning is an ongoing process in which planning activity follows revisions of prior constraint definitions and plans. Constraint revisions can apply to NSTS Constraints, SSF Hardware Constraints, and Programmatic Constraints. Plan revisions can include changes in the manifest (inter-flight plans) and changes in intra-flight plans (definition of cargo elements and placement of cargo elements in the NSTS cargo bay). When constraint/plan revisions have been made, the PAA replans, revising elements of inter- and intra-flight plans as necessary, and then validates the new plans by computing measures such as mass on each flight and comparing measures to standards defined by constraints. The PAA then reports the results of validation, which may cause the human planner to revise constraints or plan elements and rerun the PAA to generate a revised plan.

An analysis of the process description has led to definition of the following problem solving tasks to be performed by the PAA:

1. **User Review of Constraints**
   Provide support for user review of constraints, which are organized into the following categories: NSTS Constraints, e.g., volume and mass capacity, flight rate; SSF Hardware Constraints, e.g., configuration and assembly elements; and Programmatic Constraints, e.g., major program milestones and priorities. This task assumes the existence of a stored representation of constraints. To support constraints review, the PAA displays the stored representation as requested by the user.

2. **User Revision of Constraints**
   Provide support for user revision of constraints. This assumes the existence of a stored representation of constraints. The PAA provides facilities to enable the user to edit the representation.

3. **Update Constraints**
   To support user revision of constraints, the PAA changes the representation as specified by the user and propagates effects of user revisions to associated constraints.

4. **User Review of Plan Elements**
   Provide support for user review of assembly sequence plan elements, which are organized into the following categories: Inter-Flight (flight manifest); and Inter-

©ISX Corporation

Flight (cargo element definition, cargo element placement). This task assumes the existence of a stored representation of plans. To support plan review, the PAA displays the stored representation as requested by the user.

5. **User Revision of Plan Elements**
   Provide support for user revision of plan elements. This assumes the existence of a stored representation of plan elements. The PAA provides facilities to enable the user to edit the representation.

6. **Update Plans**
   To support user revision of plan elements, the PAA changes the representation as specified by the user and propagates effects of user revisions to associated plan elements.

7. **Replan**
   Given revisions of constraints and/or plan elements, replan as necessary by performing the following subtasks:
   a. Inter-Flight Planning
      Develop the flight manifest:
      i. determine the number and timing of flights
      ii. assign assembly elements to flights
   b. Intra-Flight Planning
      i. define cargo elements
      ii. place cargo elements

8. **Validate Plan**
   Given a replan, analyze and evaluate the replan by performing the following subtasks:
   a. Compute Measures, e.g., mass, volume, center of gravity (CG), the power requirement, the intra- and extra-vehicular activity requirements, and the Remote Manipulator System reach requirement
   b. Compare Measures to Standards

9. **Report Results**
   Given a user request to review constraints or plans, completion of constraint or plan element revision by the user, replanning, or validation, report results to the user by performing the following subtasks:
   a. Access Required Information from Appropriate Database(s)
   b. Format Information for Display
   c. Display Information

3.2 Task Groupings

A common heuristic in system design is that similar tasks should be performed by the same system function. This section defines four major groupings of similar tasks which serve as a starting point for the definition of major system functions.

- **User Interface**
  1. User Review of Constraints
  4. User Review of Plan Elements
  9. Report Results

- **Representation Editing**

©ISX Corporation

2. User Revision of Constraints
5. User Revision of Plan Elements

- **Representation Building**
  3. Update Constraints
  6. Update Plans

- **Plan Generation**
  7. Replan
     a. Inter-Flight Planning
     b. Intra-Flight Planning

- **Plan Validation**
  8. Validate Plan
     a. Compute Measures
     b. Compare Measures to Standards

4.0 Major System Functions

4.1 Major Functions and Requirements Specification Objects

The task groupings identified in Section 3.2 suggest the following major system functions: User Interface, Representation Editing, Representation Building, Plan Generation, and Plan Validation. These functions and associated problem-solving tasks can be mapped to the PAA objects identified in the requirements specification (see the reference in Section 2) as follows:

Major System Functions	Tasks	Requirements Objects
User Interface	1. User Review of Constraints 4. User Review of Plan Elements 9. Report Results	User Interface
Representation Editing	2. User Revision of Constraints 5. User Revision of Plans	Constraint Editor Plan Editor
Representation Building	3. Update Constraints 6. Update Plans	Constraint Builder Plan Builder
Plan Generation	7. Replan a. Inter-Flight Planning b. Intra-Flight Planning	Intra-Flight Planner Inter-Flight Planner
Plan Validation	8. Validate Plan a. Inter-Flight Planning b. Intra-Flight Planning	Value Calculators Validator

Figure 1 shows the objects and message connections identified in the requirements specification. This figure is an initial functional architecture for the PAA. It includes objects representing the major PAA functions and the data (Plan Representations and Constraint Models) and knowledge (Problem Recognition Models and Problem Solving Models) required to perform those functions.

©ISX Corporation

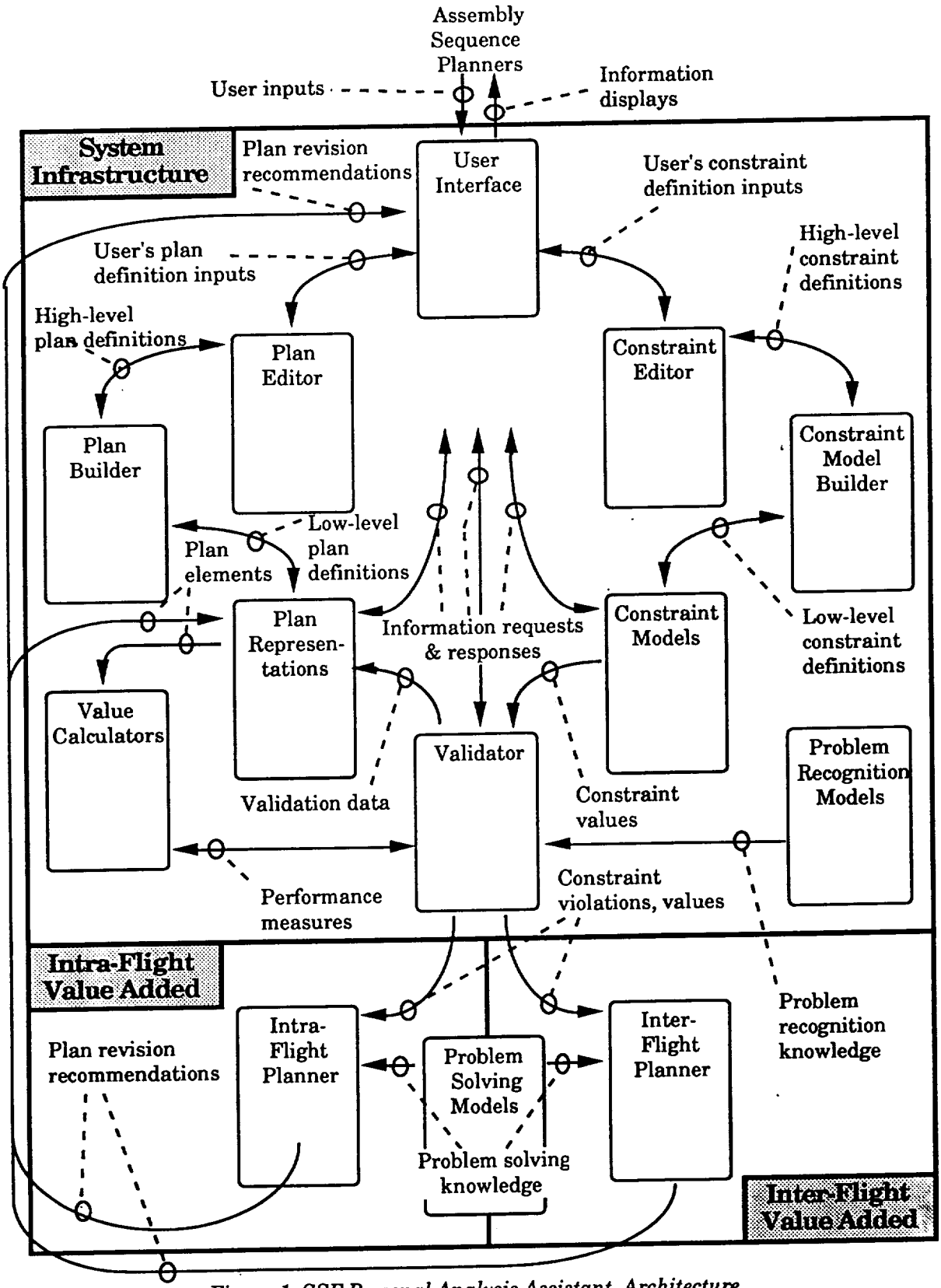


Figure 1: SSF Personal Analysis Assistant Architecture

©ISX Corporation

4.2 Generalizing the Architecture

The objective of the work performed under this Phase I SBIR is to develop requirements and a top-level design for a Manager's Assistant (MAX) system that can provide knowledge-based decision-support for SSF engineering managers. The target application is SSF assembly sequence planning, but it must not be the only application enabled by the system design. Because the PAA is only one of several "assistant" systems that can be built on the MAX framework, the objects that satisfy PAA requirements must be generalized to define the general assistant system support functions of MAX.

Figure 2 shows a generalization of this PAA software architecture that represents part of an initial top-level design for a MAX framework. The primary changes are the replacement of the Intra-Flight Planner and Inter-Flight Planner objects with a single "Planning Engine" object, the replacement of the Intra-Flight Value Added and Inter-Flight Value Added top-level objects with a single "Planner" object, and the addition of a "Development Tools" top-level object.

MAX objects form the assembly structure summarized in Figure 3. The figure uses a triangle to indicate an "is part of" relationship, e.g., the System Infrastructure is part of MAX.

The three top-level MAX software objects are defined below. Each top-level MAX software object (System Infrastructure, Planner, and Development Tools) is decomposed as shown in Figure 3. Descriptions of these lower-level objects are presented in Section 6.

Object: System Infrastructure
Description: Software that provides the capability for building representations of plans, networks of dependencies among plan objects; constraint networks describing bounds on measures; and user-specifications of constraints and analysis options.

Object: Planner
Description: A domain-independent planner design and execution environment based on ISX's ASTAIRE system, which was developed in part with support provided by a NASA ARC Phase II SBIR. ASTAIRE is a reactive planner responding to changes in situation status, e.g., constraints or plan elements. The system uses domain-dependent knowledge: Problem Recognition Models, Problem Solving Models, and Resource Capability Models. These models can be maintained and extended by user organization staff.

Object: Development Tools
Description: A development environment designed to support the creation, maintenance, and extension of assistant systems such as the PAA. Tools include editors, a user interface tool, libraries of reusable objects, a source code inspector, and a debugger. These tools can be used by user organization staff to perform system maintenance and extension of knowledge bases

©ISX Corporation

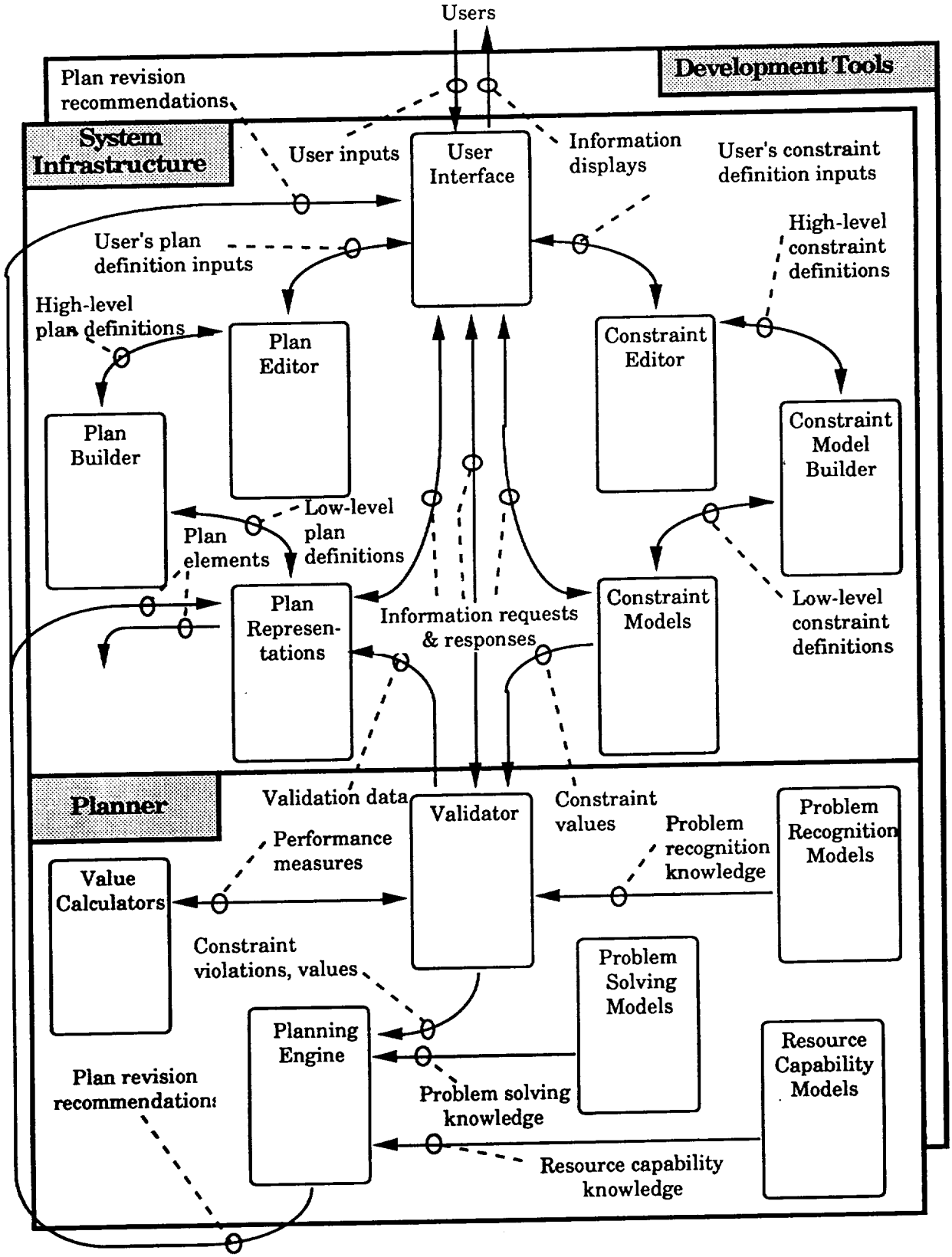


Figure 2: MAX Architecture

©ISX Corporation

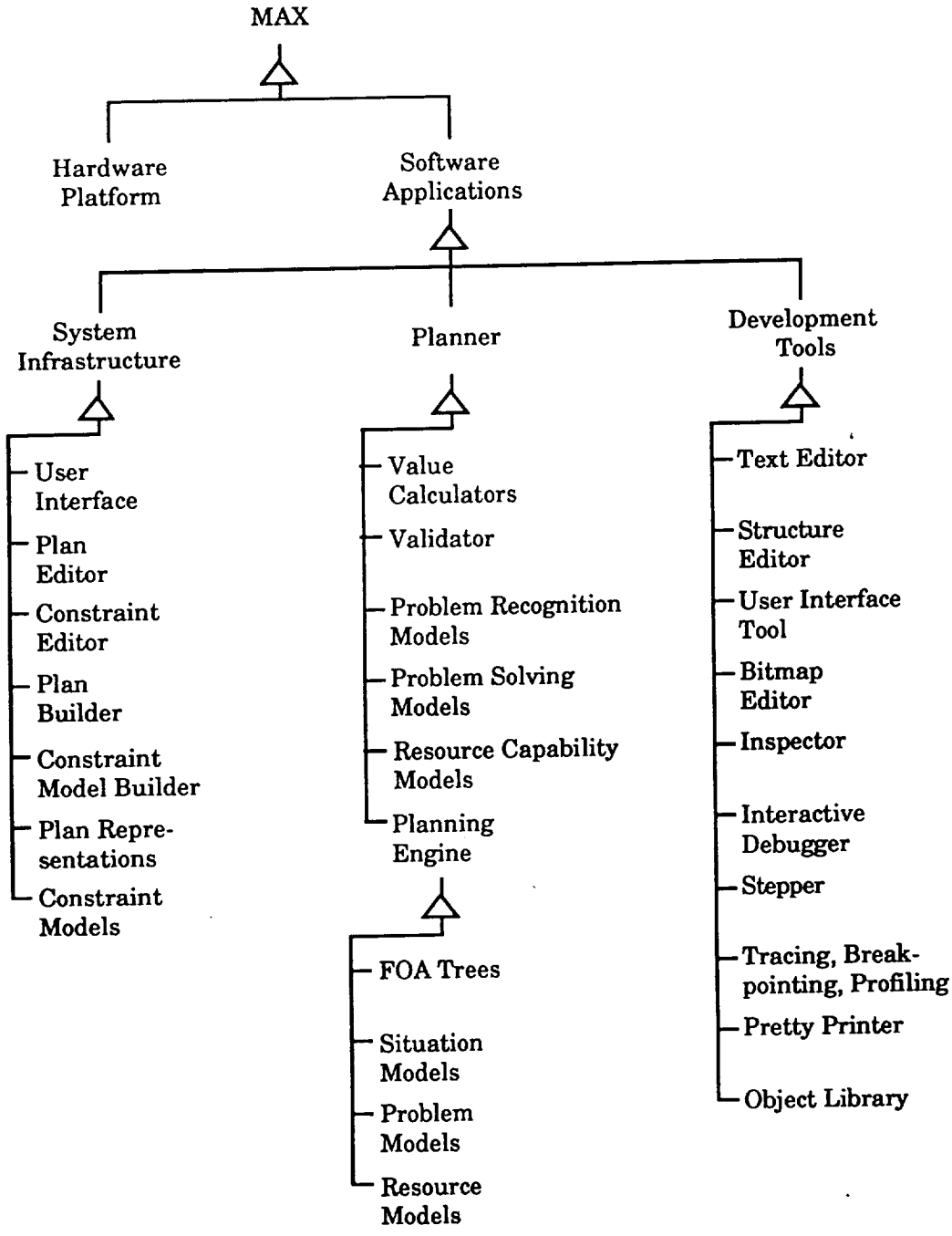


Figure 3: MAX Structure

©ISX Corporation

5.0 Technology Assessment and Selection

This section describes a preliminary assessment of alternative technologies. Technology choices are, of course, highly interdependent (an assistant system would be very helpful), and their are many interrelated constraints that must be satisfied. Assessment will continue as the design becomes more detailed, and selections will be made early in Phase II of the program.

5.1 Platform

The customer for PAA has specified a Macintosh II. For MAX, anticipated user interface and processing requirements suggests the use of a high-performance workstation such as the Macintosh II or SE/30, the TI Micro-Explorer, or the Sun 3/4. If possible, MAX should run on several different platforms with a minimum of porting. Important factors in choice of platform are performance, the availability of a development environment that provides many of the tools required for MAX Development Tools, and support for the selected knowledge and data representation(s).

5.2 Language

If knowledge and data are represented as objects, an object-oriented programming system, such as Smalltalk or CLOS, would be appropriate. Smalltalk has the advantage of being highly portable, but performance may be a problem. CLOS has good development tools, as does Smalltalk.

5.3 Development Environment

It would be highly desirable, if not required, to use a development environment that provides most of the tools specified for MAX Development Tools. Such development environments exist, e.g., ExperTelligence's Procyon Common Lisp, and it would be foolish to reinvent basic technology. The capabilities provided by the development environment is an important constraint on the selection of platforms and a language.

5.4 Representation

The highly interconnected nature of the knowledge and data in the PAA and other domains requiring the decision support of an assistant system suggests the use of object networks. The need for modularity and easy maintenance and extension of knowledge also points to the use of objects.

5.5 Database

If objects are used, an object-oriented database management system may be required for storage of plan representations, constraint representations, and planner models. ISX has experience using Gemstone™ from ServioLogic and is investigating other alternatives.

5.6 Planner

ISX's ASTAIRE, which was developed in part with support provided by a NASA ARC Phase II SBIR, provides the functionality specified for the MAX Planner. ASTAIRE uses object-based representation, and its planning is based on models of the situation, problem status, and resource status. Domain-dependent knowledge is represented in models for problem recognition, problem solving, and resource capability.

©ISX Corporation

6.0 Detailed Specifications

This section provides detailed MAX object descriptions, focusing on attributes and processing to be performed by the objects introduced in Section 4. Many of these descriptions are based on descriptions provided in the requirements specification (see the reference in Section 2). Some have been revised extensively; others have been modified only slightly. All are included for completeness. As design continues in a subsequent phase, these specifications will be extended and elaborated.

6.1 User Interface Specification

Object Name: User-Interface

Description: System Infrastructure software that provides "usable" interfaces to plan and constraint editors, representations of plan and constraint models, and the Validator object. The User-Interface object also provides input checking, a display manager, and maintenance of a User Model that enables interpretation of user input and preparation by the display manager of output meeting the user's information requirements. Initial guidelines for design of the user interface are summarized below:

- Provide users with appropriate models of the device. If a model is not provided, users will make up their own, possibly inappropriate models.
- Use affordances and constraints. Affordances suggest the range of possibilities; constraints limit the number of possibilities. Constraints can be physical, semantic, cultural, and logical.
- Use knowledge in the world, avoid knowledge in the head. Knowledge in the world acts as its own reminder. Knowledge in the head is efficient in some cases, and may be appropriate for the "power user."
- Use natural mappings; avoid labels. Appropriate use of natural mappings can minimize the use of labels. If the interface depends on labels, it may be faulty.
- Use direct manipulation. Avoid typing.
- Use generic commands to reduce the number of commands.
- State of the system must always be displayed.

Attributes:
Selection <type, location>
User-Input <type, completeness, consistency>
System-Input <type, completeness, consistency, level of abstraction>
Information-Requirements <type, level of abstraction>
User-Output <type, location, level-of-abstraction>

Processing:
Check-Selection

- check selection arguments against selection attribute constraints
- if constraints not met, return <no match> to display-manager
- if display-manager receives a <no match>, it prepares and posts a selection-failure message on the display
- if constraints are met, pass through selection to appropriate receiver

Check-User-Input

- check input arguments against input attribute constraints
- if a violation occurs, return violation identity to display-

©ISX Corporation

manager
- check input arguments against expectation defined by user-model
- if a consistency violation occurs, return violation identity to display-manager
- if display-manager receives a violation identity, it prepares and posts an input-violation message on the display
- if constraints are met, update information requirements
- if information requirements are updated, pass input and information requirements to appropriate receiver

Update-User-Model
- with user-input and expectation from user-model, define new expectation
- send new expectation to user-model

Update-Information-Requirements
- with user-input and expectation from user-model, define type and level of abstraction of information required

Prepare-Output
- check system-input against information requirements
- if a violation occurs, return system-input and information-requirements to sender
- if no violation occurs, send system-input to display-manager
- if display-manager receives system-input, it prepares and posts information on the display

6.2 Plan Editor Specification

Object Name: Plan-Editor

Description: System Infrastructure software that provides capabilities supporting viewing, selecting, and editing plan elements. This is a high-level, graphical environment for plan definition and modification.

Attributes:
Selection <type>
User-Input <type>
System-Input <element-type, completeness, consistency, level of abstraction>
Add-Element <element-type>
Change-Element <element-type, change-type>
Delete-Element <element-type>
Plan-Builder-Output <type>

Processing:
Check-Selection
- check selection arguments against selection attribute constraints
- if constraints not met, return <no match> to User-Interface object
- if constraints are met, continue to check-system-input

Check-System-Input
- check system-input arguments against system-input attribute constraints
- if a violation occurs, return violation identity to sender

Add-Plan-Element
- check user-input and system-input arguments against add-element attribute constraints
- if a violation occurs, return violation identity to sender
- if no violation occurs, send input to Plan-Builder object
- send process completion message to requester

Change-Plan-Element

©ISX Corporation

- check user-input and system-input arguments against change-element attribute constraints
- if a violation occurs, return violation identity to sender
- if no violation occurs, send input to Plan-Builder object
- send process completion message to requester

Delete-Plan-Element
- check user-input and system-input arguments against delete-element attribute constraints
- if a violation occurs, return violation identity to sender
- if no violation occurs, send input to Plan-Builder object
- send process completion message to requester

6.3 Constraint Editor Specification

Object Name: Constraint-Editor

Description: System Infrastructure software that provides capabilities supporting viewing, selecting, and editing constraint models.

Attributes:
Selection <type>
User-Input <type>
System-Input <type, completeness, consistency, level of abstraction>
Add-Constraint <constraint-type>
Change-Constraint <constraint-type>
Delete-Constraint <constraint-type>
Constraint-Model-Builder-Output <type>

Processing:
Check-Selection
- check selection arguments against selection attribute constraints
- if selection attribute constraints not met, return <no match> to User-Interface object
- if constraints are met, continue to check-system-input

Check-System-Input
- check system-input arguments against system-input attribute constraints
- if a system-input attribute constraint violation occurs, return violation identity to sender
- if no violation occurs, pass input to appropriate procedure

Add-Constraint
- check user-input and system-input arguments against add-constraint attribute constraints
- if a violation occurs, return violation identity to sender
- if no violation occurs, send input to Plan Builder
- send process completion message to requester

Change-Constraint
- check user-input and system-input arguments against change-constraint attribute constraints
- if a violation occurs, return violation identity to sender
- if no violation occurs, send input to Constraint-Model-Builder
- send process completion message to requester

Delete-Constraint
- check user-input and system-input arguments against delete-constraint attribute constraints
- if a violation occurs, return violation identity to sender
- if no violation occurs, send input to Constraint-Model-Builder
- send process completion message to requester

©ISX Corporation

6.4 Plan Builder Specification

Object Name: Plan-Builder

Description: System Infrastructure software that, given high-level Plan-Editor requests to add, change, or delete plan elements (assembly elements, cargo elements, cargo bay placement, flights, and launch manifests), generates the implementation of requests in the Plan-Representation. Implementation of plan element edits includes generation of low-level effects from high-level edits and propagation of effects throughout the affected Plan-Representation.

Attributes: Selection <type>
System-Input <type, completeness, consistency, level of abstraction>
Add-Element-Problem <element-type, problem-type, criticality>
Change-Element-Problem <element-type, change-type, problem-type, criticality>
Delete-Element-Problem <element-type, problem-type, criticality>

Processing: Check-Selection
- check selection arguments against selection attribute constraints
- if constraints not met, return <no match> to Plan-Editor object
- if constraints are met, continue to check-system-input

Check-System-Input
- check system-input arguments against system-input attribute constraints
- if a violation occurs, return violation identity to sender
- if no violation occurs, pass input to appropriate procedure

Add-Plan-Element
- identify affected Plan-Representation context
- identify proposed tasks for implementation of plan element addition
- identity problems created by proposed tasks for implementation of plan element addition (primary problems)
- identify secondary tasks created by proposed tasks for implementation of plan element addition (propagation tasks)
- identity problems created by secondary tasks for implementation of plan element addition (secondary problems)
- check primary and secondary problems against add-element-problem attribute constraints
- if a violation occurs, return violation identity and criticality to sender
- if no violation occurs, implement all tasks
- if sender confirms change, implement all tasks even with violations
- send process completion message to requester

Change-Plan-Element
- identify affected Plan-Representation context
- identify proposed tasks for implementation of plan element change
- identity problems created by proposed tasks for implementation of plan element change (primary problems)
- identify secondary tasks created by proposed tasks for implementation of plan element change (propagation tasks)
- identity problems created by secondary tasks for implementation of plan element change (secondary problems)

©ISX Corporation

- check primary and secondary problems against change-element-problem attribute constraints
- if a violation occurs, return violation identity and criticality to sender
- if no violation occurs, implement all tasks
- if sender confirms change, implement all tasks even with violations
- send process completion message to requester

Delete-Plan-Element

- identify affected Plan-Representation context
- identify proposed tasks for implementation of plan element deletion
- identity problems created by proposed tasks for implementation of plan element deletion (primary problems)
- identify secondary tasks created by proposed tasks for implementation of plan element deletion (propagation tasks)
- identity problems created by secondary tasks for implementation of plan element deletion (secondary problems)
- check primary and secondary problems against delete-element-problem attribute constraints
- if a violation occurs, return violation identity and criticality to sender
- if no violation occurs, implement all tasks
- if a violation occurs and sender confirms deletion, implement all tasks
- identity problems created by implemented tasks (primary and secondary problems)
- check primary and secondary problems against delete-element-problem attribute constraints
- if a violation occurs, return violation identity and criticality to sender
- send process completion message to requester

6.5 Constraint Model Builder Specification

Object Name: Constraint-Model-Builder

Description: System Infrastructure software that, given high-level Constraint-Editor requests to add, change, or delete constraints, generates the implementation of requests in the Constraint-Model. Implementation of constraint edits includes generation of low-level effects from high-level edits and propagation of effects throughout the affected Constraint-Model.

Attributes: Selection <type>
System-Input <type, completeness, consistency, level of abstraction>
Add-Constraint-Problem <constraint-type, problem-type, criticality>
Change-Constraint-Problem <constraint-type, problem-type, criticality>
Delete-Constraint-Problem <constraint-type, problem-type, criticality>

Processing: Check-Selection

- check selection arguments against selection attribute constraints
- if constraints not met, return <no match> to sender
- if constraints are met, continue to check-system-input

Check-System-Input

©ISX Corporation

- check system-input arguments against system-input attribute constraints
- if a violation occurs, return violation identity to sender
- if no violation occurs, pass input to appropriate procedure

Add-Constraint

- identify affected Constraint-Model context
- identify proposed tasks for implementation of constraint addition
- identity problems created by proposed tasks for implementation of constraint addition (primary problems)
- identify secondary tasks created by proposed tasks for implementation of constraint addition (propagation tasks)
- identity problems created by secondary tasks for implementation of constraint addition (secondary problems)
- check primary and secondary problems against add-constraint-problem attribute constraints
- if a violation occurs, return violation identity and criticality to sender
- if no violation occurs, implement all tasks
- if a violation occurs and sender confirms addition, implement all tasks
- identity problems created by implemented tasks (primary and secondary problems)
- check primary and secondary problems against add-element-problem attribute constraints
- if a violation occurs, return violation identity and criticality to sender
- send process completion message to requester

Change-Constraint

- identify affected Constraint-Model context
- identify proposed tasks for implementation of constraint change
- identity problems created by proposed tasks for implementation of constraint change (primary problems)
- identify secondary tasks created by proposed tasks for implementation of constraint change (propagation tasks)
- identity problems created by secondary tasks for implementation of constraint change (secondary problems)
- check primary and secondary problems against change-element-problem attribute constraints
- if a violation occurs, return violation identity and criticality to sender
- if no violation occurs, implement all tasks
- if a violation occurs and sender confirms change, implement all tasks
- identity problems created by implemented tasks (primary and secondary problems)
- check primary and secondary problems against change-element-problem attribute constraints
- if a violation occurs, return violation identity and criticality to sender
- send process completion message to requester

Delete-Constraint

- identify affected Constraint-Model context
- identify proposed tasks for implementation of constraint deletion

©ISX Corporation

- identity problems created by proposed tasks for implementation of constraint deletion (primary problems)
- identify secondary tasks created by proposed tasks for implementation of constraint deletion (propagation tasks)
- identity problems created by secondary tasks for implementation of constraint deletion (secondary problems)
- check primary and secondary problems against delete-element-problem attribute constraints
- if a violation occurs, return violation identity and criticality to sender
- if no violation occurs, implement all tasks
- if a violation occurs and sender confirms deletion, implement all tasks
- identity problems created by implemented tasks (primary and secondary problems)
- check primary and secondary problems against delete-element-problem attribute constraints
- if a violation occurs, return violation identity and criticality to sender
- send process completion message to requester

6.6 Plan Representation Specification

Object Name: Plan-Representation

Description: System Infrastructure software that stores plan representations as a network of interrelated objects. Plan-Representation also stores the results of evaluations of plan representations produced by the Validator and provides information to the User-Interface, the Plan-Builder, and Value-Calculators. Changes in plan representations are accomplished through messages sent by the Plan-Builder, the Intra-Flight-Planner, or the Inter-Flight-Planner.

Attributes: System-Input <type, completeness, information requirements>
Add-Element-Problem <element-type, problem-type, criticality>
Change-Element-Problem <element-type, change-type, problem-type, criticality>
Delete-Element-Problem <type, problem-type, criticality>
Value-Calculation <type, plan-representation-identifier>

Processing: Check-System-Input
- check system-input arguments against system-input attribute constraints
- if a violation occurs, return violation identity to sender
- if no violation occurs, pass input to appropriate procedure

Add-Plan-Element
- implement tasks for plan element addition
- identity problems created by implementation of plan element addition
- check problems against add-element-problem attribute constraints
- if a violation occurs, return violation identity and criticality to sender
- send process completion message to requester

Change-Plan-Element
- implement tasks for plan element change

©ISX Corporation

- identity problems created by implementation of plan element change
- check problems against change-element-problem attribute constraints
- if a violation occurs, return violation identity and criticality to sender
- send process completion message to requester

Delete-Plan-Element
- implement tasks for plan element deletion
- identity problems created by implementation of plan element deletion
- check problems against delete-element-problem attribute constraints
- if a violation occurs, return violation identity and criticality to sender
- send process completion message to requester

Request-Value-Calculation
- form value-calculation request using value-calculation attribute
- send value-calculation request to Value-Calculation

Record-Validation
- associate validation data received from Validation with appropriate plan element

Information-Response
- identify information meeting system-input information requirements
- get requested information
- send information to requester

6.7 Constraint Model Specification

Object Name: Constraint-Models

Description: System Infrastructure software that stores constraints as a network of interrelated objects. Constraint-Models also provides information to the User-Interface, the Constraint-Model-Builder, and the Validator. Changes in constraint models are accomplished through messages sent by the Constraint-Model-Builder.

Attributes: System-Input <type, completeness, information requirements>
Add-Constraint-Problem <constraint-type, problem-type, criticality>
Change-Constraint-Problem <constraint-type, change-type, problem-type, criticality>
Delete-Constraint-Problem <type, problem-type, criticality>

Processing: Check-System-Input
- check system-input arguments against system-input attribute constraints
- if a violation occurs, return violation identity to sender
- if no violation occurs, pass input to appropriate procedure

Add-Constraint
- implement tasks for constraint addition
- identity problems created by implementation of constraint addition
- check problems against add-element-problem attribute constraints
- if a violation occurs, return violation identity and criticality to sender

©ISX Corporation

- send process completion message to requester

Change-Constraint
- implement tasks for constraint change
- identity problems created by implementation of constraint change
- check problems against change-element-problem attribute constraints
- if a violation occurs, return violation identity and criticality to sender
- send process completion message to requester

Delete-Constraint
- implement tasks for constraint deletion
- identity problems created by implementation of constraint deletion
- check problems against delete-element-problem attribute constraints
- if a violation occurs, return violation identity and criticality to sender
- send process completion message to requester

Information-Response
- identify information meeting system-input information requirements
- get requested information
- send information to requester

6.8 Value Calculator Specification

Object Name: Value-Calculator

Description: Planner software that identifies and computes performance measures appropriate to the plan representation information sent by the Plan-Representation object. Measures are domain-dependent measures of effectiveness and values of constrained variables. This object provides information to the Validator.

Attributes: System-Input <type, completeness>
Report-Form <type, parameters, plan-representation>

Processing: Check-System-Input
- check system-input arguments against system-input attribute constraints
- if a violation occurs, return violation identity to sender
- if no violation occurs, pass input to appropriate procedure

Identify-Measures
- select measures appropriate for plan representation contained in value-calculation-request received from Plan-Representations

Get-Plan-Representation
- request plan data from Plan-Representation

Compute-Measures
- compute identified measures on plan data supplied by Plan-Representation

Report-Measures
- form report using Report-Form attributes
- send computed performance measures with plan representation identifier to Validator

©ISX Corporation

6.9 Validator Specification

Object Name: Validator

Description: Planner software that compares values of performance measures provided by Value-Calculator to standards defined by Constraint-Models and determines whether measures violate constraints using knowledge provided by Problem-Recognition-Models. Provides information to the User-Interface, Plan-Representations, and the Planning Engine.

Attributes: System-Input <type, completeness>
Report-Form <type, parameters, plan-representation>

Processing: Check-System-Input
- check system-input arguments against system-input attribute constraints
- if a violation occurs, return violation identity to sender
- if no violation occurs, pass input to appropriate procedure

Get-Constraints
- identify constraints relevant to performance measures provided by the Value-Calculator
- get identified constraints

Validate
- compare performance measures with standards defined by constraints
- send comparisons to Problem-Recognition-Models for evaluation

Report-Results
- develop report using response from Problem-Recognition-Models and Report-Form attributes
- send report with plan representation identifier to User-Interface, Plan-Representations, Intra-Flight-Planner, and Inter-Flight-Planner.

6.10 Problem Recognition Model Specification

Object Name: Problem-Recognition-Models

Description: Planner software that is a knowledge-based server for the Validator. The object evaluates comparisons of performance measures and standards defined by constraints provided by the Validator to determine whether measures violate constraints. It provides results of the evaluation to the Validator.

Attributes: System-Input <type, completeness>

Processing: Check-System-Input
- check system-input arguments against system-input attribute constraints
- if a violation occurs, return violation identity to sender
- if no violation occurs, pass input to appropriate procedure

Identify-Knowledge
- identify knowledge relevant to evaluating measure-constraint comparisons provided by the Validator

Evaluate
- execute evaluation by applying identified knowledge to measure-constraint comparisons
- send results to the Validator

©ISX Corporation

6.11 Problem Solving Model Specification

Object Name: Problem-Solving-Models
Description: Planner software that is a knowledge-based server for the Planning-Engine. Problem-Solving-Models contains knowledge about how to solve plan problems in the domain. The object uses information in the solution-request provided by the Planning-Engine to identify and prioritize potential solutions. It provides results to the Planning-Engine.
Attributes: System-Input <type, completeness>
Processing: Check-System-Input
- check system-input arguments against system-input attribute constraints
- if a violation occurs, return violation identity to sender
- if no violation occurs, pass input to appropriate procedure

Identify-Knowledge
- identify knowledge relevant to identifying and prioritizing solutions
- execute identification and prioritization
- send results to requester

6.12 Resource Capability Model Specification

Object Name: Resource-Capability-Models
Description: Planner software that is a knowledge-based server for the Planning-Engine. Resource-Capability-Models contains knowledge about the capabilities of resources in the domain. The object uses information in the solution-request provided by the Planning-Engine to identify and prioritize resources applicable to a problem solution. It provides results to the Planning-Engine.
Attributes: System-Input <type, completeness>
Processing: Check-System-Input
- check system-input arguments against system-input attribute constraints
- if a violation occurs, return violation identity to sender
- if no violation occurs, pass input to appropriate procedure

Identify-Knowledge
- identify knowledge relevant to identifying and prioritizing resources
- execute identification and prioritization
- send results to requester

6.13 Planning Engine Specification

Object Name: Planning-Engine
Description: Planner software that, given constraint violation problems identified by the Validator, problem solutions suggested by Problem-Solving-Models, and prioritized resources provided by Resource-Capability-Models, selects and implements solutions. Solutions are implemented by developing and executing task plans producing required changes to the Plan-Representation. Has the following parts: FOA Trees, Situation Models, Problem Models, and Resource Models. Provides information to the User-Interface and Plan-Representations.

©ISX Corporation

Attributes:
Selection <type>
System-Input <type, completeness, consistency, level of abstraction>
Solution-Request <problem-type>
Resource-Request <problem-type>
Add-Element-Problem <element-type, problem-type, criticality>
Change-Element-Problem <element-type, change-type, problem-type, criticality>
Delete-Element-Problem <element-type, problem-type, criticality>

Processing:
Check-System-Input
- check system-input arguments against system-input attribute constraints
- if a violation occurs, return violation identity to sender
- if no violation occurs, pass input to appropriate procedure

Update-Situation-Model
- using Validator input, update the Situation-Model

Update-Problem-Model
- using Validator input, identify problem to be solved
- define a Focus of Attention (FOA) object for the problem to be solved
- define goal objects for the FOA object
- write FOA object into Problem-Model

Update-Resource-Model
- given selected resources, define resource objects for goal object(s)
- write the FOA object into Resource-Model

Get-Solutions
- prepare solution-request using solution-request attributes
- send solution-request to Problem-Solving-Models

Select-Solution
- evaluate solutions provided by Problem-Solving-Models to identify a recommended solution
- send recommendation to User-Interface
- if user confirms change, implement all tasks
- if user does not confirm change, select and recommend another solution
- if no solution can be selected, stop

Get-Resources
- prepare resource-request using resource-request attributes
- send resource-request to Resource-Capability-Models

Select-Resources
- evaluate resources provided by Resource-Capability-Models to identify a recommended resources for solution
- send recommendation to User-Interface
- if user confirms change, implement all tasks
- if user does not confirm change, select and recommend another solution
- if no solution can be selected, stop

Implement-Addition-Tasks
- identify affected Plan-Representation context
- identify proposed tasks for implementation of plan element addition
- identity problems created by proposed tasks for implementation of plan element addition (primary problems)
- identify secondary tasks created by proposed tasks for

©ISX Corporation

implementation of plan element addition (propagation tasks)
- identity problems created by secondary tasks for implementation of plan element addition (secondary problems)
- check primary and secondary problems against add-element-problem attribute constraints
- if a violation occurs, return violation identity and criticality to sender
- if no violation occurs, implement all tasks
- if sender confirms change, implement all tasks even with violations
- send process completion message to requester

Implement-Change-Tasks
- identify affected Plan-Representation context
- identify proposed tasks for implementation of plan element change
- identity problems created by proposed tasks for implementation of plan element change (primary problems)
- identify secondary tasks created by proposed tasks for implementation of plan element change (propagation tasks)
- identity problems created by secondary tasks for implementation of plan element change (secondary problems)
- check primary and secondary problems against change-element-problem attribute constraints
- if a violation occurs, return violation identity and criticality to sender
- if no violation occurs, implement all tasks
- if sender confirms change, implement all tasks even with violations
- send process completion message to requester

Implement-Delete-Tasks
- identify affected Plan-Representation context
- identify proposed tasks for implementation of plan element deletion
- identity problems created by proposed tasks for implementation of plan element deletion (primary problems)
- identify secondary tasks created by proposed tasks for implementation of plan element deletion (propagation tasks)
- identity problems created by secondary tasks for implementation of plan element deletion (secondary problems)
- check primary and secondary problems against delete-element-problem attribute constraints
- if a violation occurs, return violation identity and criticality to sender
- if no violation occurs, implement all tasks
- if a violation occurs and sender confirms deletion, implement all tasks
- identity problems created by implemented tasks (primary and secondary problems)
- check primary and secondary problems against delete-element-problem attribute constraints
- if a violation occurs, return violation identity and criticality to sender
- send process completion message to requester

©ISX Corporation

6.14 Development Tools

This section presents brief descriptions of required MAX Development Tools. Because many MAX Development Tools will be provided by commercial development environment, the tools have not been defined in as much detail as other MAX objects. Detailed specifications will be developed for tools that cannot be obtained commercially.

Object Name: Text Editor
Description: A full-featured text editor, integrated with all other tools, that will allow users to edit text and programs without leaving the development environment.

Name: Structure Editor
Description: A graphic structure editor, integrated with all other tools, that will allow users to edit plan and constraint representations by moving and changing properties of icons.

Name: User Interface Tool
Description: A tool or integrated set of tools for designing and developing user interfaces. This should include high-level support for graphics, fonts, windows, menus, and mice; user-defined stream/window managers; functions and macros for high-level, machine-independent graphics and user interface programming.

Name: Bitmap Editor
Description: A bitmap editor, integrated with all other tools, that will allow mouse-driven editing and creation of pictures, cursors, brushes, and textures.

Name: Inspector
Description: A browser tool for dynamic examination and modification of data structures.

Name: Debugger
Description: An interactive debugger supporting full access to runtime stack and environment. The Debugger should be called when the flow of execution is interrupted, and it should allow inspection and modification of values stored on the execution stack. All programming tools should be available within the Debugger execution context.

Name: Stepper
Description: The Stepper allows single-step debugging of code. Features include single step, skipping of subexpressions, aborting, and entering the Debugger.

Name: Tracing, Breakpointing, & Profiling
Description: Tracing a function causes diagnostic information to be printed every time the function is called or returns. Setting a breakpoint on a function gives the option of entering the Debugger every time that function is called or returns. Profiling a function accumulates timing information on the function.

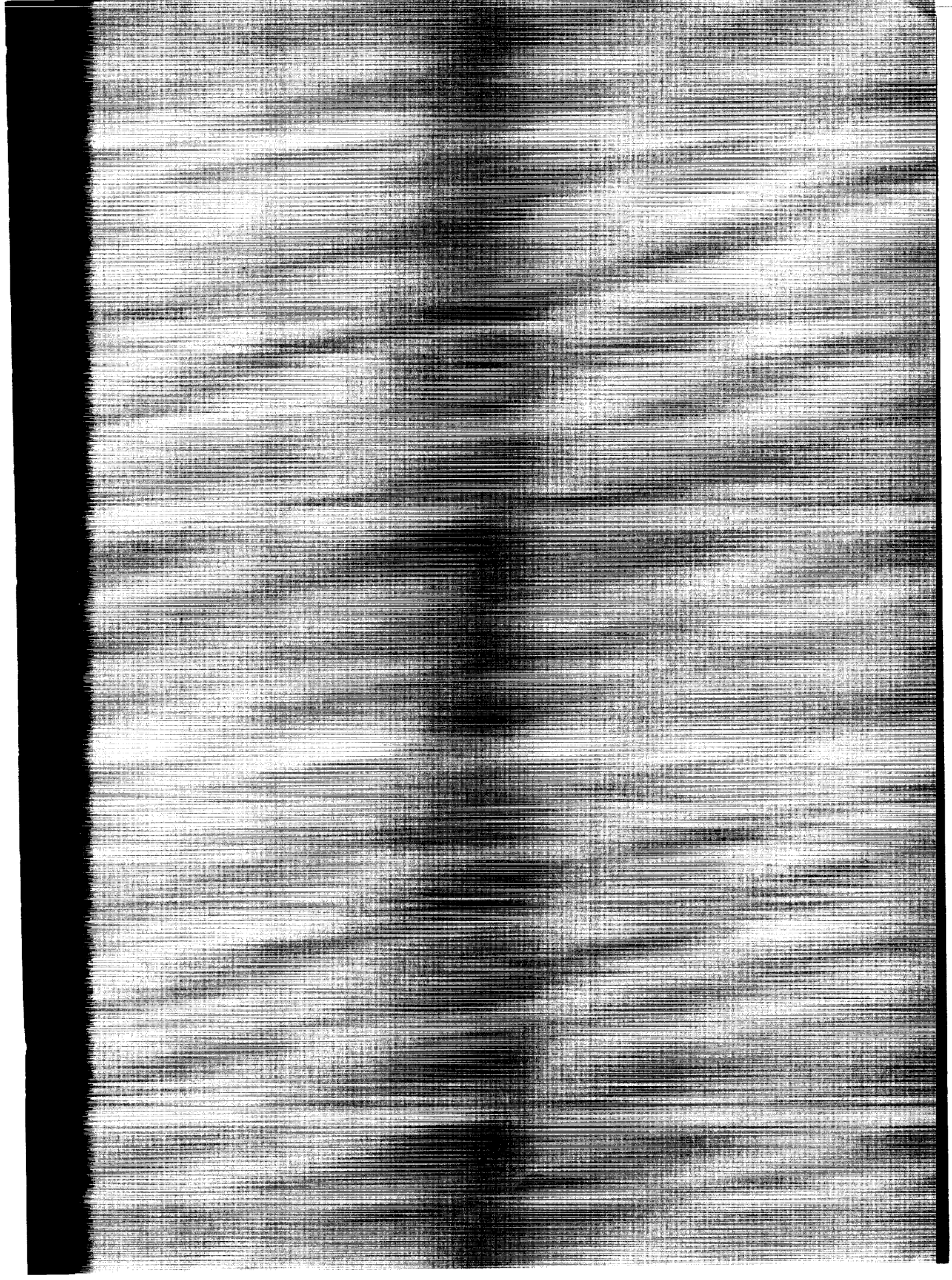
Name: Pretty Printer
Description: The Pretty Printer produces formatted output of data structures and

©ISX Corporation

source code. Features include user-extensible print macros to control printing and support for multi-font and variable-width characters.

Name: Object Library
Description: A collection of reusable, editable System Infrastructure and Planner objects supporting the revision, maintenance, and extension of existing systems and the development of new systems.

©ISX Corporation



Assembly Sequence Planning Personal Analysis Assistant

User's Reference Guide

Version 1.0b

Prepared by

ISX Corporation
501 Marin Street, Suite 214
Thousand Oaks, CA 91360

February 14, 1991

Funded By

NASA Ames Research Center
NASA Research Announcement Contract NAS2-13296

Table Of Contents

Section **Page #**

Copyright © 1991, ISX Corporation

NOTES TO THE USER

(1) It is assumed that the user is familiar with the Apple Macintosh™ operating system and general style of application operation and behavior.

(2) The use of **bold** typeface is merely intended to convey special emphasis to the reader and is used throughout this document.

(3) To "**mouse**" on an object is to place the cursor over the object and depress the mouse button.

(4) "**Double-clicking**" on an object is to place the cursor over the object and depress the mouse button twice.

(5) When editing a data field it is necessary to "**tab**", that is press the tab key, out of the field to actually change the value of the data field.

Copyright © 1991, ISX Corporation

1.0 Overview

The Personal Analysis Assistant (PAA) application is designed to automate the tasks of assembly sequence planning for construction of Space Station Freedom (SSF). These tasks include mission launch scheduling, SSF assembly element mission allocation, Space Transportation System (STS) cargo bay loading, and SSF program milestone creation and tracking.

Hardware and software system requirements are established in Section 2.0. Section 3.0 explains system installation. The top level PAA window is detailed in Section 4.0. Sections 5.0 through 13.0 cover the major PAA dialogs. The proposed Cargo Bay Editor design is discussed in Appendix A, Trunnion data format is presented in Appendix B, and known system bugs are listed in Appendix C.

2.0 System Requirements

2.1 Hardware Requirements

The PAA application requires one of the Macintosh II™ family of computers, or a SE.30™, with 8 megabytes of memory and a 19" monitor (color optional). There should be a minimum of 7.5 megabytes of hard disk space available for installation and 5 megabytes for operation.

2.2 Software Requirements

The PAA application requires Finder 6.1.5 and System 6.0.5, or later. The PAA also requires Procyon Common Lisp v2.1.5d runtime application (provided).

NOTE: Due to the size of the PAA application it is recommended that it be run under the Finder only, as opposed to the Multifinder.

3.0 System Installation

3.1 Disk Contents

The PAA comes on four (4) double-sided (800K) 3.5 inch diskettes. These disks contain a compressed version of the PAA application and its associated files. The files have been compressed with an application called **DiskDoubler™**. An extract-only application called **DDExpand™** has been provided to decompress the PAA files. Disks **PAA.1**, **PAA.2**, **PAA.3**, and **PAA.4** contain the compressed files, which has intern been split to fit on the distribution diskettes. Disk **PAA.4** also contains **DDExpand™**. The five distribution files are shown in Figure 3.1.

Figure 3.1 - Distribution Files

Copyright © 1991, ISX Corporation

3.2 Installation Procedure

To install the PAA, copy the 5 files mentioned in Section 3.1 onto your hard disk. Double-click on the **PAA.dd.1** file. The **DDExpand™** application will combine the 4 PAA.dd.# files into a file named **PAA.dd**, removing the 4 PAA.dd.# files. Then it will expand the **PAA.dd** file into a folder named **PAA**. This folder contains the PAA application and its associated files. Remove the **PAA.dd** and the **DDExpand™** files. Double-click on the PAA folder. Its contents are listed Figure 3.2.

Figure 3.2 - Installed PAA Files

To start the PAA application, double-click on the **PAA** icon.

Copyright © 1991, ISX Corporation

4.0 PAA Window

When the PAA application is started the top level window opens, see Figure 4.1. This window displays available orbiters, along the left vertical axis, versus mission launch dates, along the horizontal axes. The top horizontal axis shows the range of dates displayed in the mission display field. The bottom horizontal axis shows the range of dates for the entire program. Missions are displayed as rectangles in the mission display field and are labeled with the appropriate mission name. The window resize box is located in the lower right corner. Window operations are described in Section 4.1 and menu commands are discussed in Section 4.2.

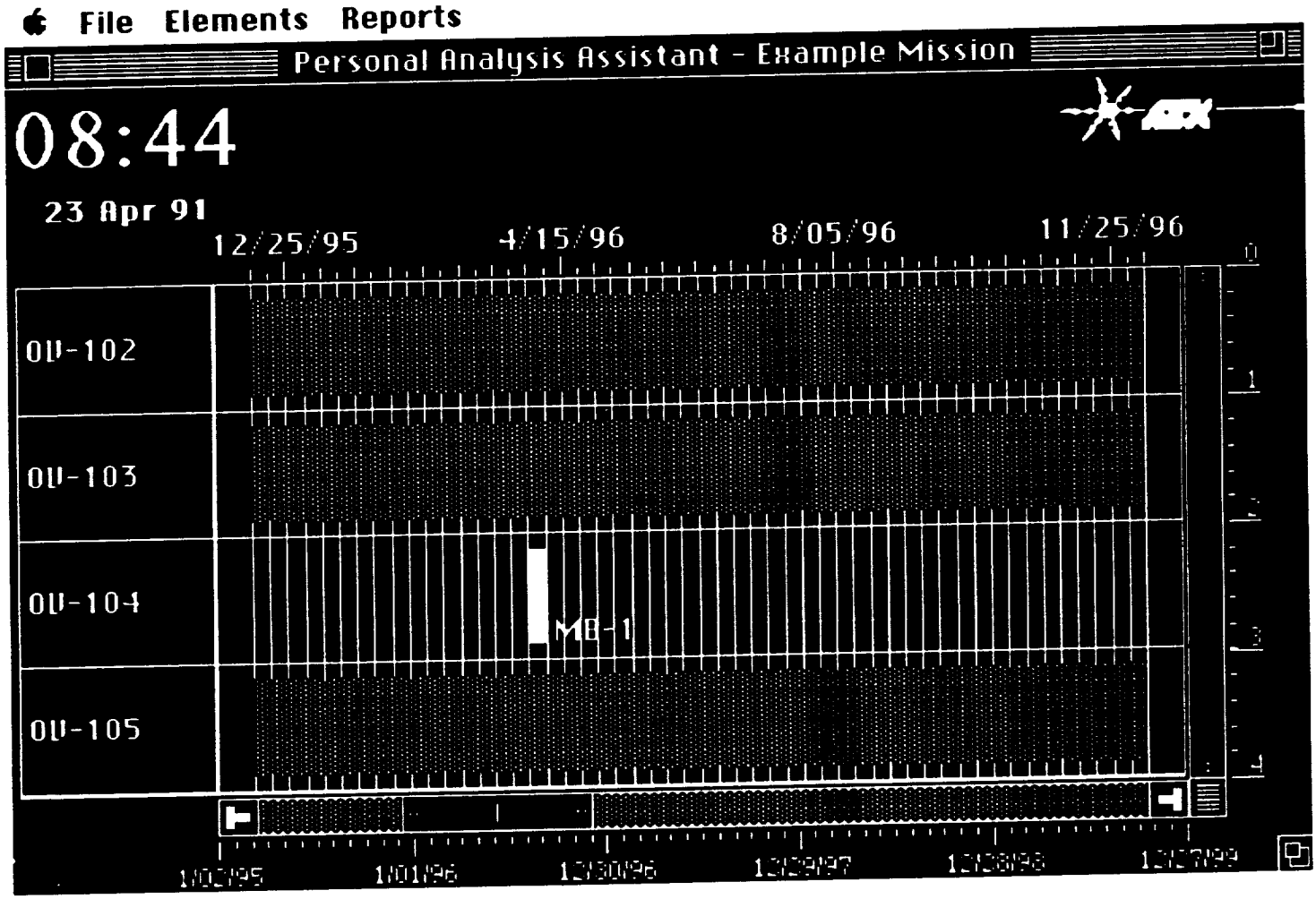


Figure 4.1 - Top Level PAA Window

4.1 Window Operations

4.1.1 Missions

Missions are displayed as rectangles and are positioned in the horizontal row of the orbiter which will carry out the mission, and with their left edge in the column which aligns with the mission launch date, to the nearest week. The Mission Dialog for a particular mission is opened by mousing on the desired mission. The mission's Mission Dialog may also be opened from the Open Mission Dialog, see Section 8.0.

Copyright © 1991, ISX Corporation

4.1.2 Scrolling

The mission display field may be scrolled both horizontally and vertically, as required, by using the scroll bars at the bottom and the right of the display field, respectively. Mousing in the scroll bar, but to either side of the scroll rectangle, will cause the display field to scroll in that direction, one year per click for the horizontal bar, and one orbiter at a time for the vertical one. Mousing in the scroll buttons at the extreme ends of the bar will cause the display field to scroll all the way to the display field end of the selected button.

4.1.3 Scaling

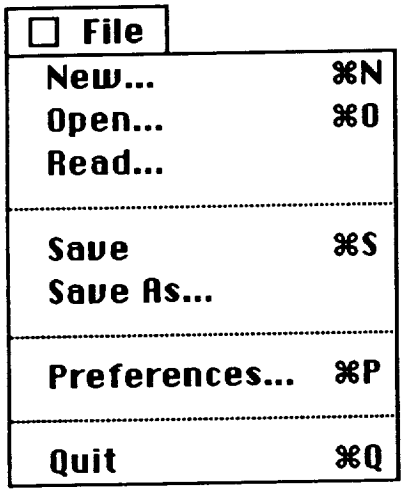
A unique feature of the PAA scroll bars is that they also enable the mission display field to be scaled, as well as scrolled. To scale the display field move the cursor to the end of the scroll rectangle nearest the desired scale direction, for example, over the << or the >> for the horizontal bar. While holding down the **Command** key (⌘), press and hold the mouse button, and drag the mouse in the desired direction. This will cause the scroll rectangle to lengthen, or shorten. A number will appear in the scroll rectangle and change as the mouse is dragged. This number represents the number of weeks, for the horizontal bar, which will be displayed in the mission display field, the default is 52 weeks. Releasing the mouse button will cause the mission display field to be redrawn to the indicated scale. The dates displayed at the upper edge of the display field will change accordingly *(Bug: There is a display bug which prevents the dates from displaying correctly above 72 weeks).*

Mousing on the button in the lower right of the field, at the intersection of the scroll bars, will cause the mission display field to be scaled as to be fully displayed. To display a region of the display field, move the mouse to the upper left corner of the desired region. While holding down the **Control** key, press the mouse button and drag the mouse toward the lower right of the display field. A rectangle will be drawn as the mouse is dragged. When the mouse button is released, the display field will be scaled and redrawn to the region bounded by the rectangle.

Copyright © 1991, ISX Corporation

4.2 Menu Commands

4.2.1 File Menu



The File menu commands are described below:

New... ⌘N
Create a new SSF program run. If the presently loaded run has not been saved the user will be prompted to save.

Open... ⌘O
Open an existing SSF program run. If the presently loaded run has not been saved the user will be prompted to save.

Read...
Read a tab delimited run data file, typically an Excel™ spreadsheet . This command merges the incoming data with the existing run.

Save ⌘S
Save the existing run to disk with the current name.

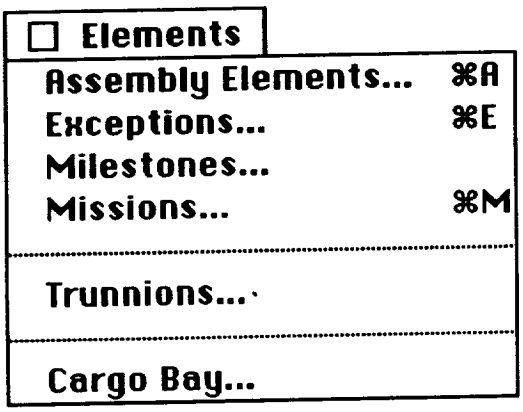
Save as...
Save the existing run to disk with a new name.

Preferences... ⌘P
Open the Preferences Dialog, see Section 13.0.

Quit... ⌘Q
Exit the PAA application. If the presently loaded run has not been saved the user will be prompted to save.

Copyright © 1991, ISX Corporation

4.2.2 Elements Menu



The **Elements** menu commands are described below:

Assembly Elements... ⌘A
Open the Assembly Element Dialog, see Section 5.0.

Exceptions... ⌘E
Open the Exceptions Dialog, see Section 6.0.

Milestones...
Open the Milestone Dialog, see Section 7.0.

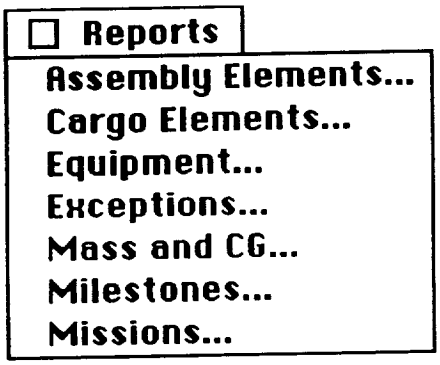
Missions... ⌘M
Open the Open Mission Dialog, see Section 8.0.

Trunnions...
Read in new C language format trunnion data, see Appendix B.

Cargo Bay...
Display the Phase II Cargo Bay Editor, see Appendix A.

Copyright © 1991, ISX Corporation

4.2.3 Reports Menu



The **Reports** menu commands are described below:

Assembly Elements...
Write a tab delimited file of all assembly element data.

Cargo Elements...
Write a tab delimited file of all mission cargo element data.

Equipment...
Write a tab delimited file of all mission equipment data.

Exceptions...
Write a tab delimited file of all exception data.

Mass and CG...
Write a tab delimited file of all mission mass and center of gravity data.

Milestones...
Write a tab delimited file of all program milestone data.

Missions...
Write a tab delimited file of all mission data.

Copyright © 1991, ISX Corporation

5.0 Assembly Element Dialog

The **Assembly Element Dialog** allows the user to view, create, edit, or delete SSF assembly elements, see Figure 5.1. The dialog consists of the assembly element list, three action buttons, **New**, **Delete**, and **Done**, and the assembly element data fields. Assembly elements are displayed with their numbers, names, and mission affiliation.

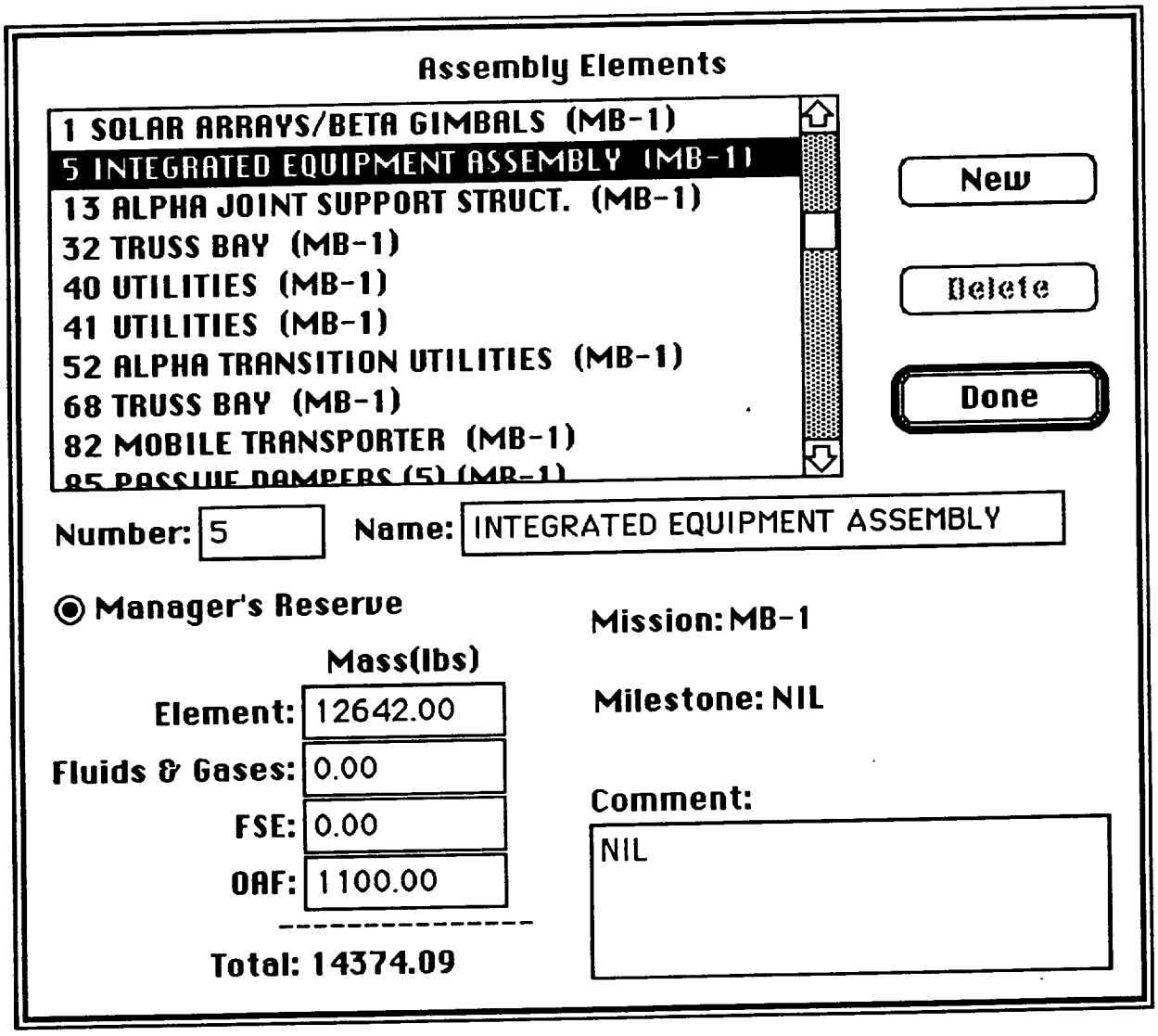


Figure 5.1 - Assembly Element Dialog

To create a new assembly element, mouse on the **New** button. An assembly element named "untitled" will be created and added to the assembly element list. To edit an assembly element, select it from the element list. The assembly element data fields will be populated with element's data. Select the desired values to change, tabing between fields. Note that an assembly element must have a unique number, except for elements with the number zero (used for elements which currently have no valid number). In order to delete an element, it cannot belong to a mission. An element can only be removed from a mission in the **Cargo Assembly Dialog**, see Section 11.0. To delete an element that does not belong to a mission, select it and mouse on the **Delete** button.

Copyright © 1991, ISX Corporation

6.0 Exceptions Dialog

The **Exceptions Dialog** displays a listing of all program exceptions, see Figure 6.1. This dialog is intended for information only and does not affect any data in the current run. The dialog shown in the figure lists a single Mass Margin exception of 503.69 lbs against the mission named "MB-1" *(NOTE: The present version of the PAA application also tracks both Forward and Aft Center of Gravity exceptions)*. When viewing of the dialog is completed, mouse on the **Done** button or press the carriage return key.

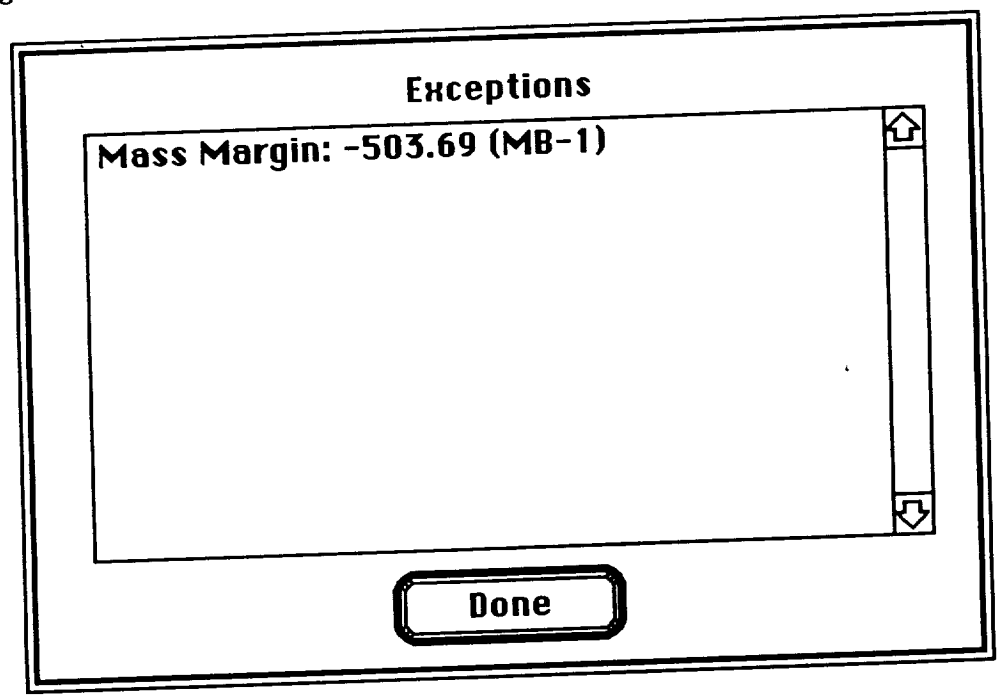


Figure 6.1 - Exceptions Dialog

The **Exceptions Dialog** is also displayed when the **Mission Dialog** is opened and if the selected mission has exceptions against it.

Copyright © 1991, ISX Corporation

7.0 Milestone Dialog

The **Milestone Dialog** allows the user to view, create, edit, or delete program milestones, see Figure 7.1. The dialog consists of the available assembly elements list (those elements not already associated with a milestone), the milestones list, the milestone assembly elements list, the add and remove assembly element icons, the add and delete milestone buttons, the milestone name field, the milestone date popup menus, and the **Done** button.

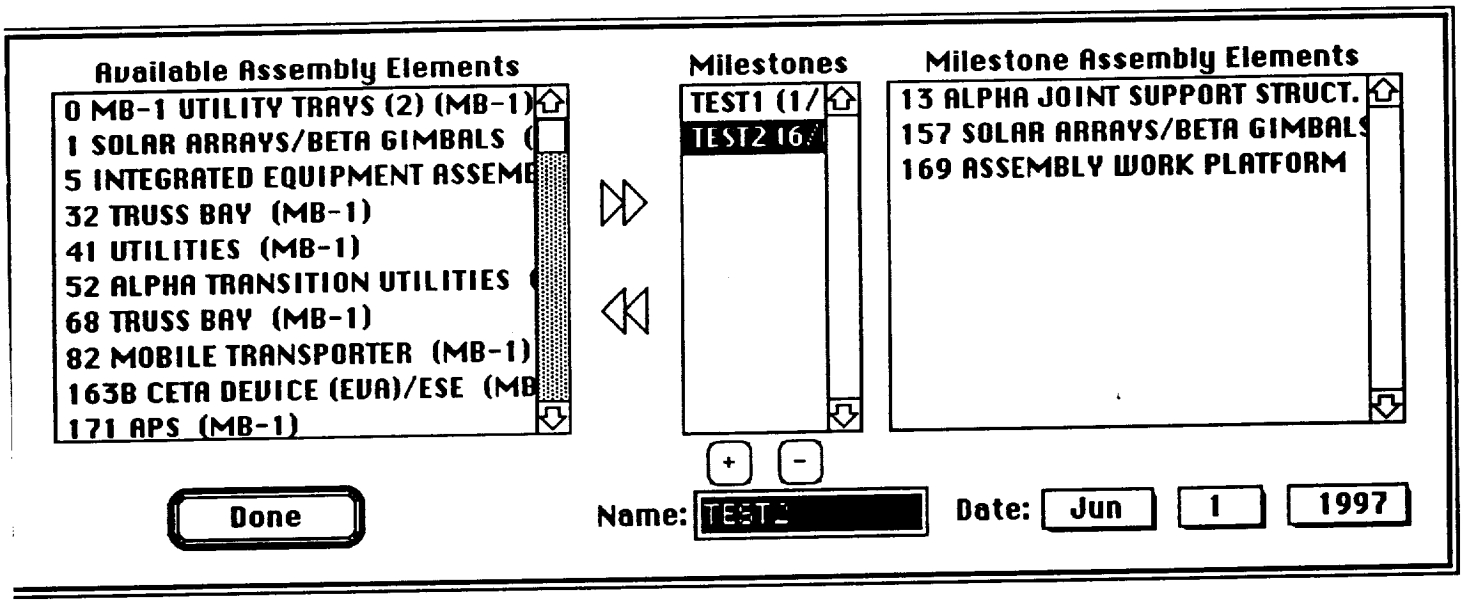


Figure 7.1 - Milestone Dialog

To create a milestone mouse on the add milestone button, the square "+" button under the milestone list. A milestone named "untitled" will be displayed on the milestone list. To edit the milestones name or date, select it from the milestone list, its name and date will be displayed in the milestone date field and popups. Also any assembly elements which belong to this milestone will be displayed in the milestone assembly list. To add an assembly element to a milestone, select both the element and the milestone. Then mouse on the add assembly element icon, the right pointing double arrowhead. The element will be removed from the available element list and added to the milestone element list. To remove an element, select both the milestone and the element and mouse on the left pointing double arrowhead. To remove a milestone, select it and mouse on the remove milestone button, the square "-" button. Note that a milestone must be empty of assembly elements before it can be deleted. When finished, mouse on the **Done** button.

Copyright © 1991, ISX Corporation

8.0 Open Mission Dialog

The **Open Mission Dialog** allows the user to open or delete an existing mission, or create a new one, see Figure 8.1. The dialog consists of the mission list, and the **Open**, **New**, **Delete**, and **Done** buttons.

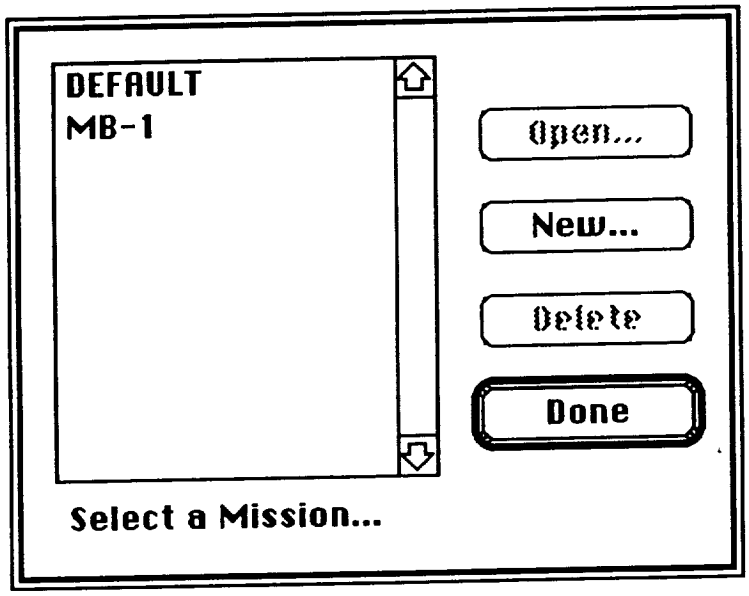


Figure 8.1 - Open Mission Dialog

To open an existing mission, select the mission and mouse on the **Open** button. This will cause the dialog to close and the **Mission Dialog** to open, see Section 9.0. To create a new mission, mouse on the **New** button. A dialog will be displayed to name the new mission. On acceptance of the name the **Mission Dialog** will be opened the to new mission. Note that the attributes of mission named "**Default**" are inherited by any newly created mission. Only missions which have no assembly elements may be deleted. To remove a mission's assembly elements go the **Cargo Assembly Dialog**, see Section 11.0. To delete a mission which has no assembly elements, select the mission and mouse on the **Delete** button.

Copyright © 1991, ISX Corporation

9.0 Mission Dialog

The **Mission Dialog** allows the user to edit an existing mission, see Figure 9.1. The dialog consists of various mission attribute fields and popups, mass and center of gravity data summary displays, **Equipment**, **Contents**, **Cargo Bay**, **Print**, **Save**, and **Cancel** buttons.

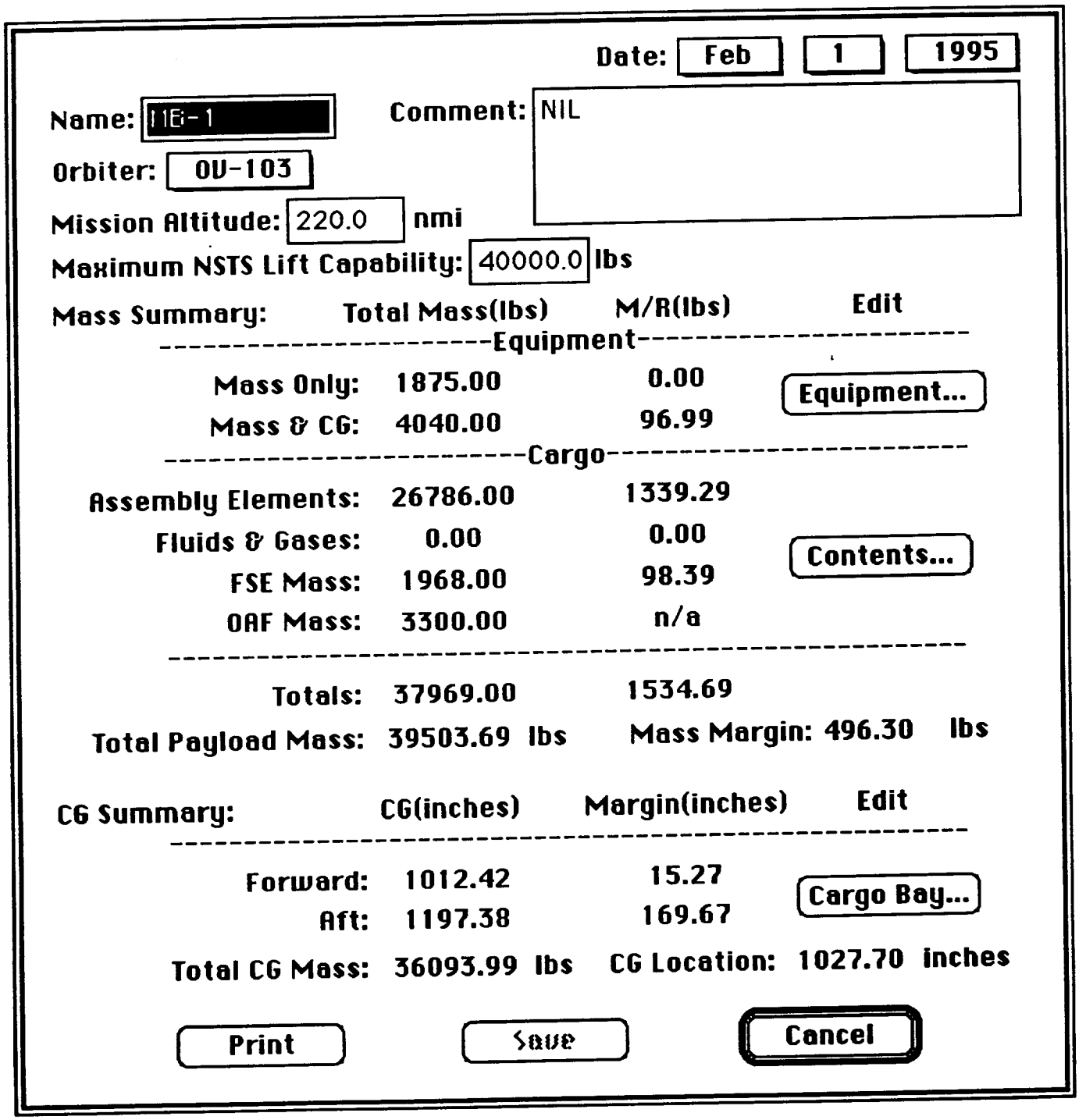


Figure 9.1 - Mission Dialog

Copyright © 1991, ISX Corporation

Mission attributes may be changed as described in the preceding sections, Note that the mission name must be unique. To edit the mission's equipment elements mouse on the **Equipment** button, see Section 10.0. To add or remove assembly elements from the mission, create or delete cargo elements, or populate existing cargo elements, mouse on the **Contents** button, see Section 11.0. To configure existing cargo elements or load the cargo bay, mouse on the **Cargo Bay** button. To print a tab delimited mission report, mouse on the **Print** button. To save the mission in the current run, mouse on the **Save** button. To cancel the edit session and return to the PAA top level window, mouse on the **Cancel** button or press the carriage return key.

Copyright © 1991, ISX Corporation

10.0 Equipment Dialog

The **Equipment Dialog** allows the user to view, create, edit, or delete mission equipment, see Figure 10.1. The dialog consists of the equipment element list, three action buttons, **New**, **Delete**, and **Done**, and the equipment element data fields.

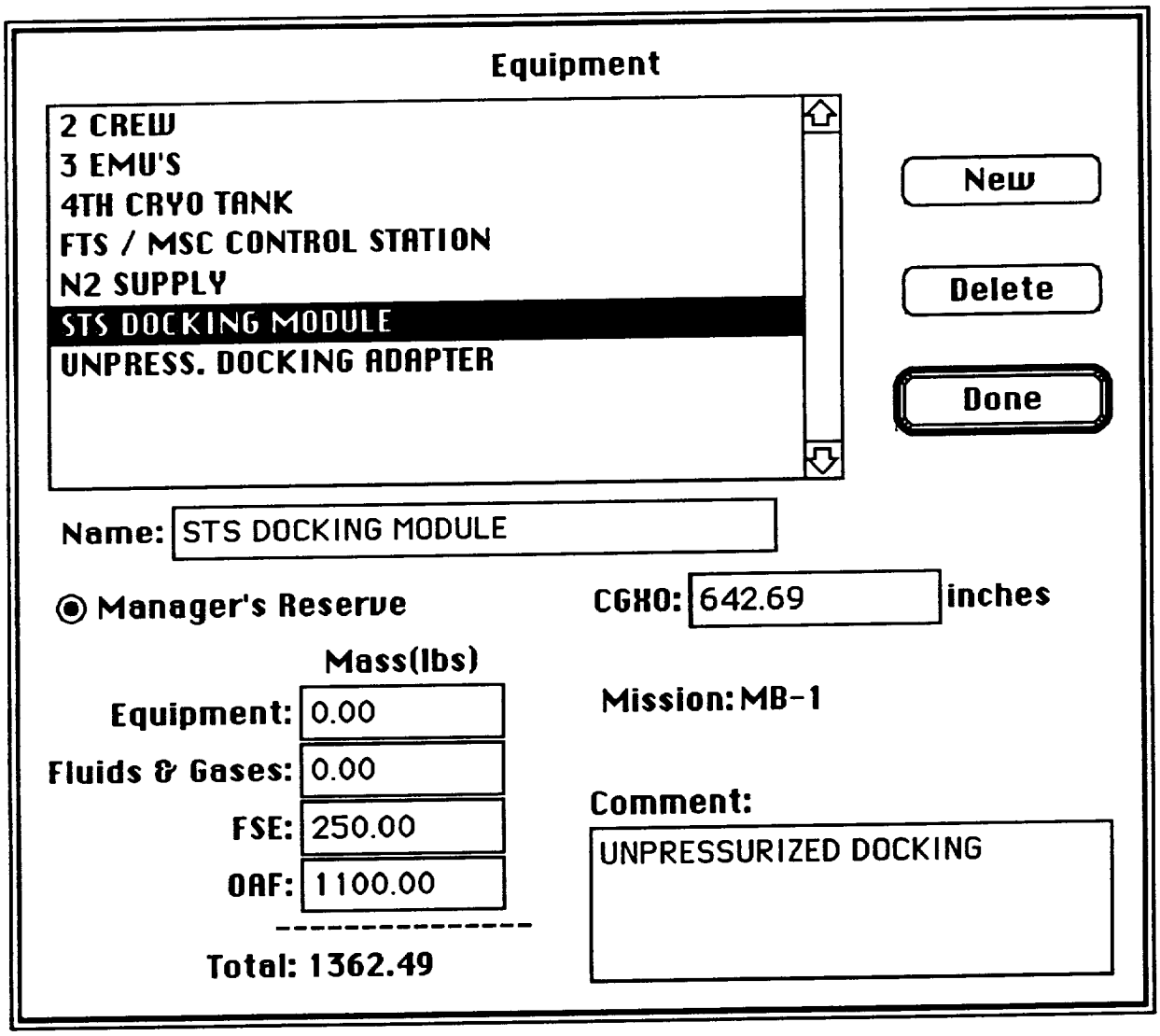


Figure 10.1 - Equipment Dialog

To create an equipment element, mouse on the **New** button. An equipment element named "untitled" will be created and added to the equipment element list. To edit an equipment element, select it from the element list. The equipment element data fields will be populated with element's data. Select the desired values to change, tabing between fields. To delete an element, select it and mouse on the **Delete** button.

Copyright © 1991, ISX Corporation

11.0 Cargo Assembly Dialog

The **Cargo Assembly Dialog** allows the user to add or remove assembly elements from the current mission, create or delete cargo elements, or add or remove mission assembly elements from existing cargo elements, see Figure 11.1. The dialog consists of the available assembly element list, the mission assembly element list, the mission cargo element list, the cargo element assembly element list, the cargo element name field, the add and remove mission assembly element icons, the add and remove cargo element assembly element icons, the new and delete cargo element buttons, and the **Done** button.

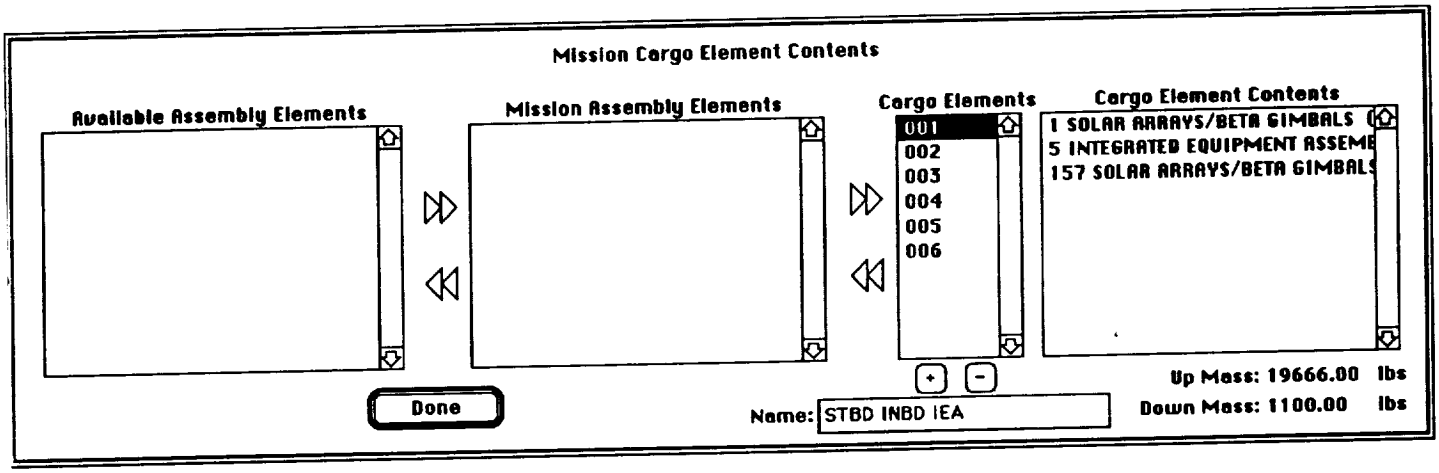


Figure 11.1 - Cargo Assembly Dialog

To create a cargo element mouse on the add cargo element button, the square "+" button under the cargo element list. A cargo element named "untitled" will be created and an automatically generated id number displayed on the cargo element list. To edit the element's name, select the element, its name will filled into the name field. To delete an element, select it and press the "-" key. Note that a cargo element must be empty before it can be deleted.

To add an assembly element to the mission, select the element from the available assembly element list. Mouse on the add assembly element icon, the right pointing double arrowhead between the available assembly element list and the mission assembly elements list. To remove an element from the mission, select the element from the mission assembly element list, Mouse on the remove assembly element icon, the left pointing double arrowhead between the available assembly element list and the mission assembly element list.

To add an assembly element to an existing cargo element, select the assembly element from the mission assembly element list and the cargo element from the cargo element list. Mouse on the add cargo element assembly element icon, the right pointing double arrowhead between the mission assembly element list and the mission cargo element list. To remove an assembly element from a cargo element, select the assembly element from the cargo element assembly element list, Mouse on the remove cargo element assembly element icon, the left pointing double arrowhead between the mission assembly elements list and the cargo element assembly element list. Note that to modify a cargo element, by adding or deleting assembly elements, it not be loaded into the cargo bay. To remove a loaded cargo element from the cargo bay, use the **Cargo Bay Dialog**, see Section 12.0

To exit the dialog, mouse on the **Done** button.

Copyright © 1991, ISX Corporation

12.0 Cargo Bay Dialog

The **Cargo Bay Dialog** allows the user to edit various attributes of existing cargo elements, load and unload the orbiter cargo bay, and alter several orbiter cargo bay constraints, see Figure 12.1. The dialog consists of three main regions. The upper region displays all existing cargo elements and the fields for editing their attributes. The middle region displays the orbiter cargo bay constraint variables. The lower region displays the trunnion, cargo element loading, and orbiter payload data summaries.

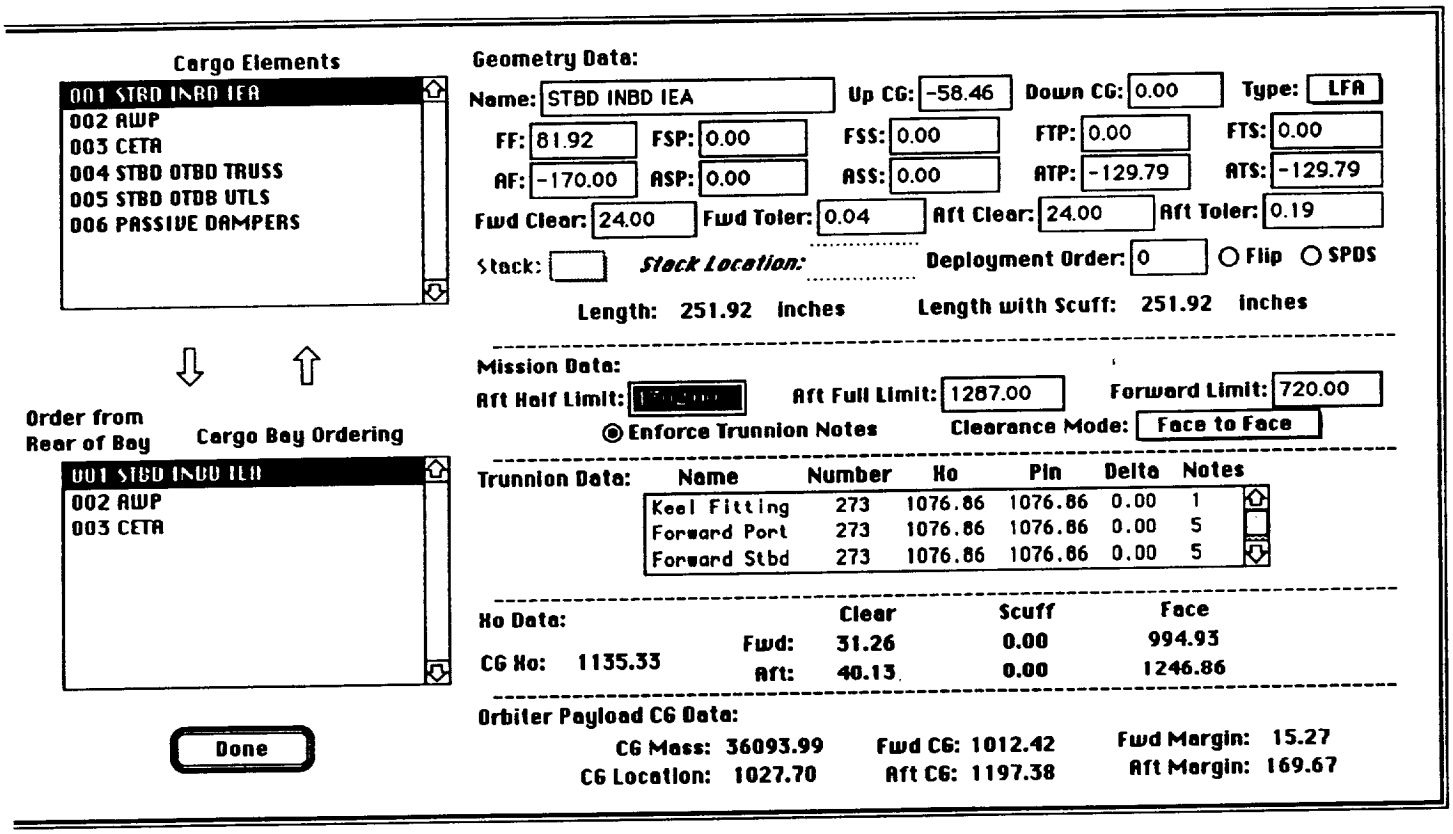


Figure 12.1 - Cargo Bay Dialog

To edit an existing cargo element's attributes, select the element from the cargo elements list. The geometry data fields will be filled with the element's data. Edit the data by tabing between fields and selecting choices from the popup menus and the radio buttons. Note that only those cargo elements that are not loaded into the cargo bay may be edited.

The mission data, orbiter cargo bay constraints, may be edited at any time. However, if any cargo elements are loaded into the cargo bay the user will be warned that changes to these valued may invalidate the current cargo bay configuration. It is recommended that the bay be unloaded prior to changing these values.

Copyright © 1991, ISX Corporation

To load the cargo bay, select a cargo element from the cargo elements list. Mouse on the load cargo element icon, the downward pointing arrow. The element will be loaded into the bay and be displayed in the cargo bay list. To unload an element, select it from the cargo bay list. Mouse on the unload cargo element icon. The element will be unloaded and removed from the cargo bay list. Note that if the element to be unloaded is has loaded elements forward of it in the cargo bay, they will also be unloaded.

To exit the dialog, mouse on the **Done** button.

Copyright © 1991, ISX Corporation

13.0 Preferences Dialog

The **Preferences Dialog** allows the user to edit the global system values, see Figure 13.1. The dialog consists of various data fields, which may be edited by selecting the values and tabing between fields. To save the new values, mouse on the **Save** button. To cancel any changes, mouse on the **Cancel** button. *(Bug: Changing the Start or End year will not cause the top level PAA window to change its start or end years until the system has been saved and reloaded).*

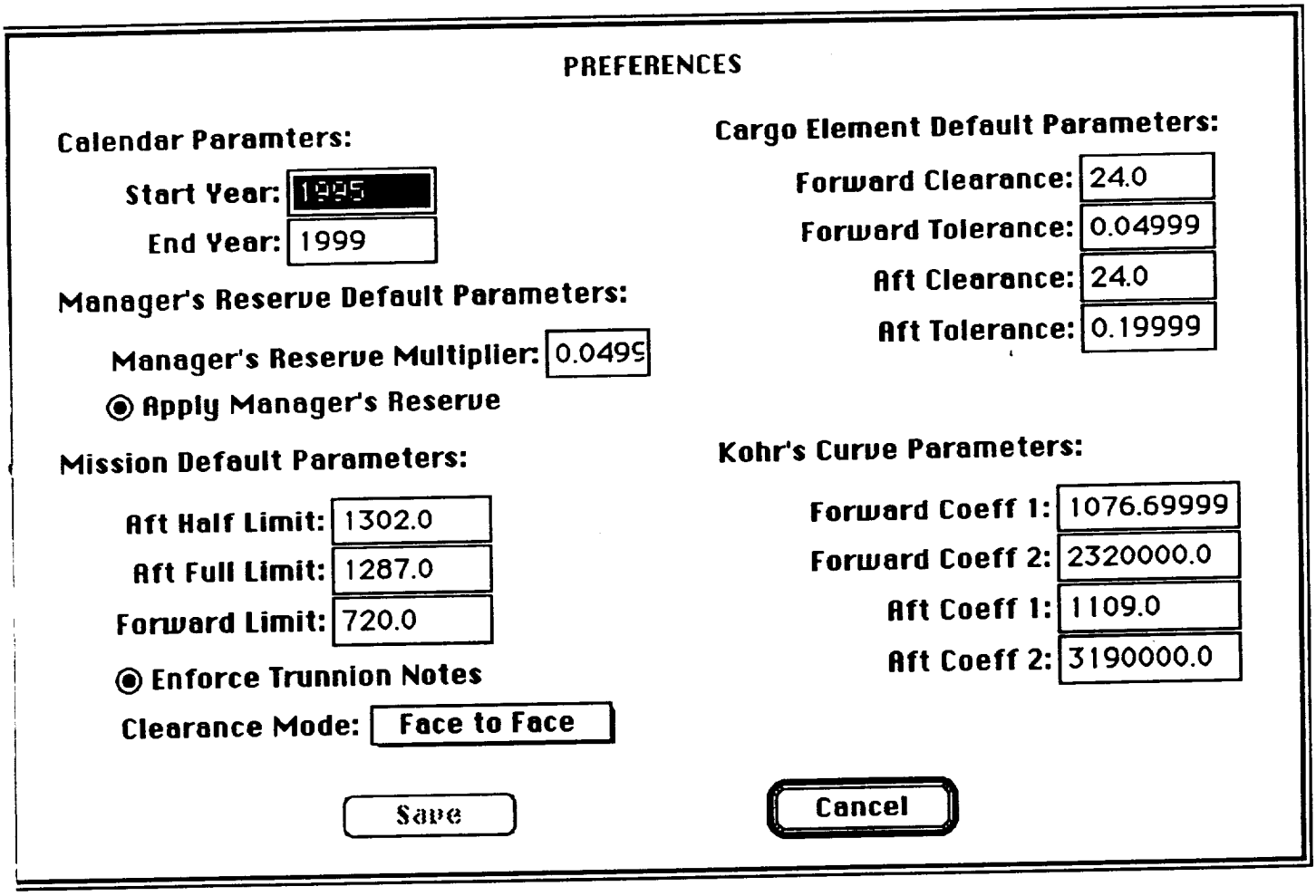


Figure 13.1 - Preferences Dialog

Copyright © 1991, ISX Corporation

Appendix A - Cargo Bay Editor

Cargo Bay Editor, see Figure A-1, is intended to be implemented in Phase II of the program and to replace the Cargo Bay Dialog shown in Section 12.0. This approach utilizes a user-friendly, graphically oriented, interface for defining a mission's cargo element geometry and placement in the cargo bay.

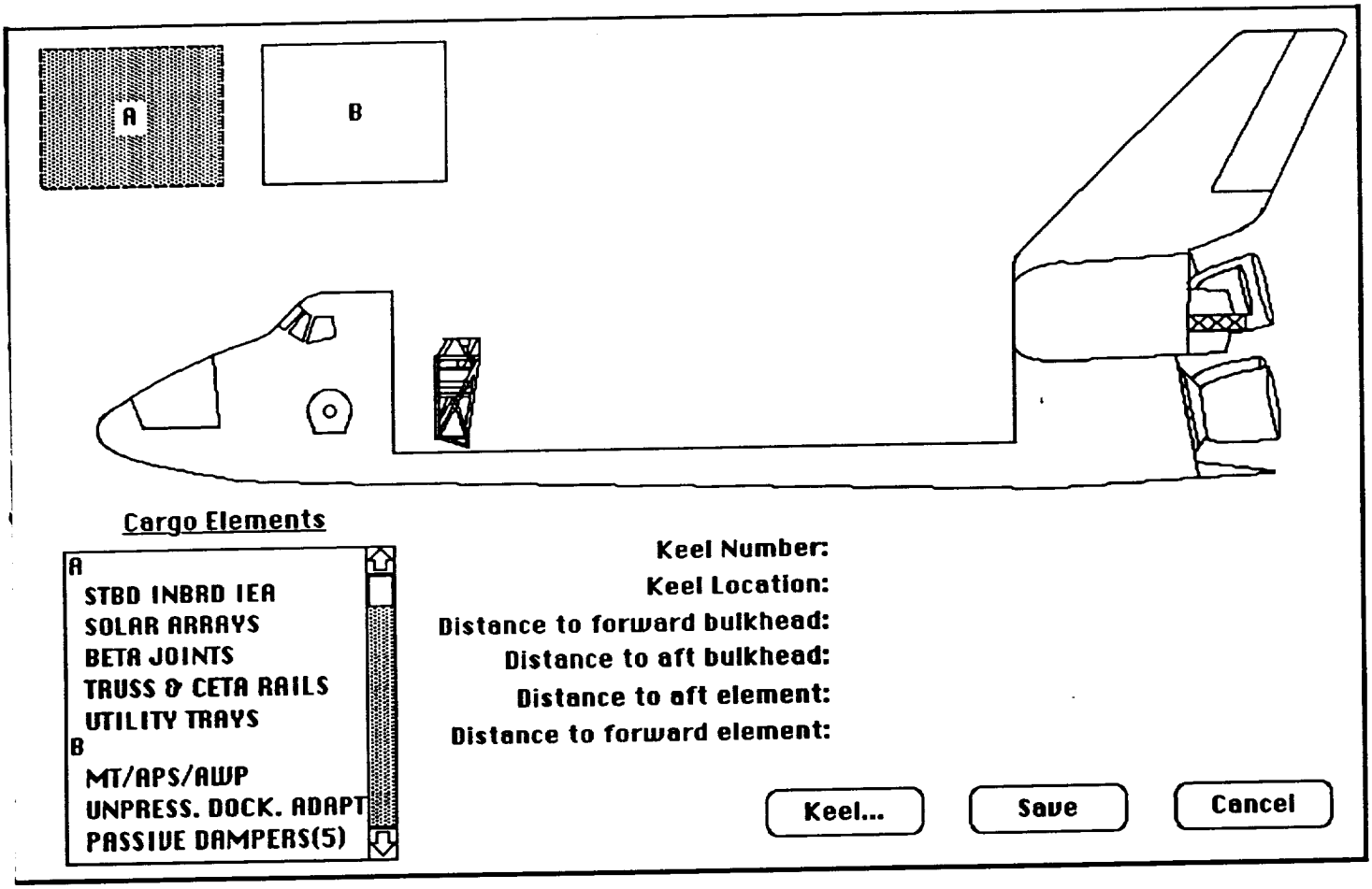


Figure A-1 - Cargo Bay Editor

The upper left corner of the editor displays the envelopes for this mission's cargo elements. The elements are labeled according to the cargo element ids specified in the Mission Dialog, see Section 9.0. An element which is grayed-out signifies that its geometry has not been defined by the user. Selecting this element will cause a dialog to appear which will invite the user to specify the element's geometric attributes, including its envelope, keel pin and trunnion point locations, cg location, and whether it is a full or half height element. Saving this dialog will cause the user to be returned to the Cargo Bay Editor and the selected element to become white. A white element may be dragged into the cargo bay for placement. The user selects the element, and while holding down the mouse button, drags it toward the cargo bay. When the element reached the bottom of the bay, it will "click" into place at the nearest keel pin/trunnion location which matches the element's configuration. As the user slides the element along the bottom of the bay, the element will continue to "click" into the next acceptable location. As the element approaches another element, or the forward or aft cargo bay bulkhead, an alert is displayed when the element violates the buffer distance constraint (defined in the Cargo Bay Editor Preferences dialog, not shown). The user then has the option of overriding the violated constraint or returning the element to the last

Copyright © 1991, ISX Corporation

acceptable location. The user may overlap elements, to provide for the case where an element has a convex face which complements a concave face of a adjacent element. The user may also "reverse" the orientation of an element, such that the element's forward face becomes the aft face and visa versa. Directly below the cargo bay are a series of text fields which provide dynamic information about the state of the currently selected element and the center of gravity margin as determined by the Kohr's memo constraint curve (the parameters of this curve may be edited in the aforementioned preferences dialog). A description of each cargo element is shown in the lower left corner of the editor while the "Save", "Cancel", and "Keel..." buttons are located in the lower right. Pressing the "Keel..." button displays a dialog which enables the user to alter the configuration of the keel pin/trunnion locations.

Copyright © 1991, ISX Corporation

Appendix B - Trunnion Data

The Trunnion data file must be in the following format:

```
155, 612.73, 13, -1, -1, -1, -1,
156, 616.67,  0, -1,  3,  7,  9,
157, 620.60,  0, -1,  3,  7,  9,
158, 624.53, -1, -1,  3,  7,  9,
159, 628.47, -1, -1,  3,  7,  9,
160, 632.40, -1, -1,  3,  7,  9,
163, 644.20, -1, -1,  3,  7, -1,
164, 648.13, -1, -1,  3,  7, -1,
165, 652.07, -1, -1,  3,  7, 10,
                  •
                  •
                  •
313, 1234.20, -1, -1,  9, -1, -1,
314, 1238.13, -1, -1,  9, -1, -1,
315, 1242.07, -1, -1,  9, -1, -1,
316, 1246.00, -1, -1,  3,  9, -1,
323, 1273.53, -1, -1,  3, -1, -1
```

Where the fields are from left to right the trunnion number, the Xo location, the three keel fitting note fields, and the two longeron fitting note fields.

Copyright © 1991, ISX Corporation

Appendix C - Known Bugs

(1) When the top level PAA window is scaled beyond 72 weeks the upper date scale is not displayed properly.

(2) Changing the Start or End year will not cause the top level PAA window to change its start or end years until the system has been saved and reloaded.

(3) Changing the name a PAA data file with the Finder will not result in its name changing within the PAA application.

(4) Many lists within the PAA application do not respond to double-clicking on a selection.

(5) When a mission data attribute is changed in the Cargo Bay Dialog and cargo elements are loaded into the cargo bay, it may take several attempts to exit the dialog with the Done button.

(6) When attempting to change the number of an assembly element in the Assembly Element Dialog and the new number already exists in the system, an error dialog will be diaplayed when the user attempts to exit the dialog.

Copyright © 1991, ISX Corporation

Index

Copyright © 1991, ISX Corporation